

# **Development Solutions**

80387 Support Library Reference Manual



#### NOTATIONAL CONVENTIONS

This manual uses the following notation:

mac and maer are prefixes written in lowercase except in

examples. The Decimal Conversion Library and Common Elementary Function Library

routines begin with mac and maer.

respectively. These prefixes are lowercase to make each routine's name more readable.

are expressed as a..b and are inclusive: a and Ranges

b are part of the range.

ST..ST(7) denote the 80387 stack elements. ST and

ST(0) are equivalent representations for the 80387 stack top; the seven remaining stack elements are ST(1)..ST(7), respectively.

represent elements of the set of real numbers. x and v

> i represents an element of the set of integers.

z and w represent elements of the set of complex

numbers.

A represents an angle expressed in radians.

H suffix or (Hex) indicates a value expressed in hexadecimal.

Otherwise, values are expressed in decimal.

indicates one or more lines of code omitted from an example. The omitted lines are application-specific rather than essential to an

understanding of the example.

# 80387 SUPPORT LIBRARY REFERENCE MANUAL

Order Number: 455497-001

REV.	REVISION HISTORY	DATE
-001	Original Issue.	12/88
•		

In the United States, additional copies of this manual or other Intel literature may be obtained from:

Literature Department Intel Corporation 3065 Bowers Avenue Santa Clara, CA 95051

In locations outside the United States, obtain additional copies of Intel documentation by contacting your local Intel sales office. For your convenience, international sales office addresses are printed on the last page of this document.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's Software License Agreement, or in the case of software delivered to the government, in accordance with the software license agreement as defined in FAR 52.227-7013.

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel Corporation.

Intel Corporation retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

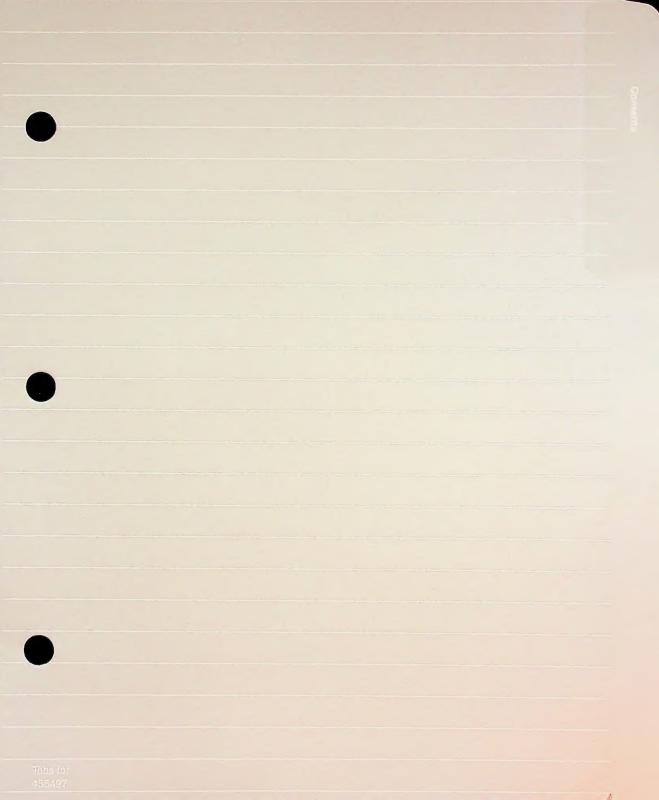
The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

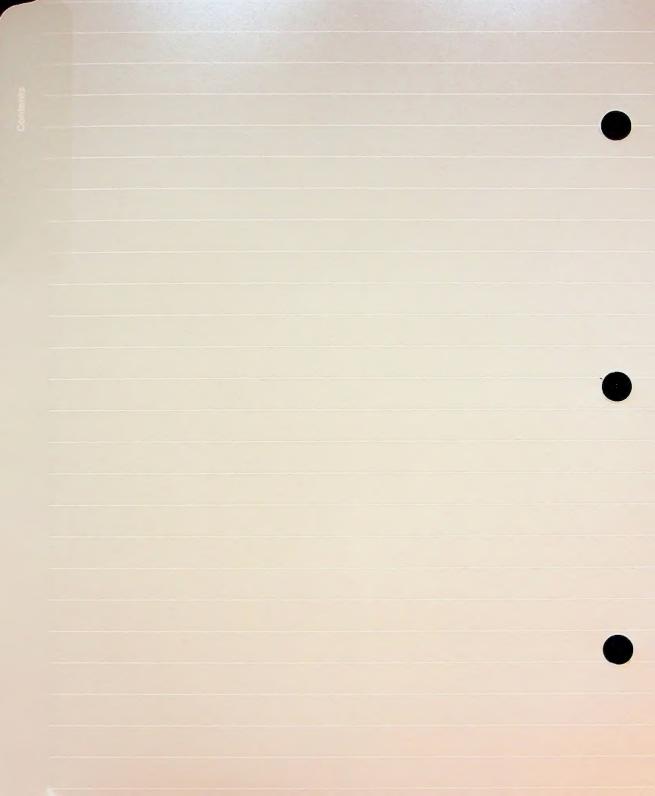
Above	iLBX	Intellink	MICROMAINFRAME	Ripplemode
BITBUS	im	IOSP	MULTIBUS	RMX/80
COMMputer	iMDDX	iPAT	MULTICHANNEL	RUPI
CREDIT	iMMX	iPDS	MULTIMODULE .	Seamless
Data Pipeline	Inboard	iPSC	ONCE	SLD
ETOX	Insite	IRMK	OpenNET	SugarCube
FASTPATH	Intel	iRMX	OTP	UPI
Genius	intel	ISBC	PC BUBBLE	VLSiCE
A	Intel376	iSBX	Plug-A-Bubble	376
i	Intel386	iSDM	PROMPT	386
12ICE	intelBOS	iSXM	Promware	386SX
ICE	Intel Certified	KEPROM	QueX	387
ICEL	Intelevision	Library Manager	QUEST	387SX
iCS	Inteligent Identifier	MAPNET	Quick-Erase	4-SITE
iDBP	Inteligent Programming	MCS	Quick-Pulse Programming	
iDIS	Intellec	Megachassis		

IBM is a registered trademark of International Business Machines Corporation.

IBM Personal System/2 is a registered trademark of International Business Machines Corporation.

Copyright @ 1988, Intel Corporation, All Rights Reserved





# Contents

Pre	face		viii
Ch	apter	1 Numerics Support and 80387 Initialization	
1.1	80387	Support Library Overview	1-1
	1.1.1	80387 Support Library Configurations	1-2
	1.1.2	80387 Support Library Linkage	1-2
1.2	Initia	lization Library Overview	1-3
	1.2.1	INIT87 and INITFP Routines	1-3
	1.2.2	Declaring Initialization Routines in ASM386 Programs	1-4
Ch	apter	2 Decimal Conversion Library	
2.1	Libra	ry Overview	2-1
	2.1.1	Declaring DC387 Routines in ASM386 Programs	2-2
		Declaring DC387 Parameters and Data Structures	
		DC387 Results	
		DC387 Stack Requirements	
		DC387 Register Usage	
		80387 Control Word Settings	
		DC387 Numeric Exceptions	
2.2		nal Conversion Library Routines	
		Summary of DC387 Routines	
		How to Read the DC387 Reference Pages	
		DC387 Reference	

# **Chapter 3 Common Elementary Function Library**

3.1	Libra	ry Overview3-1
	3.1.1	Declaring CL387 Routines in ASM386 Programs 3-2
	3.1.2	CL387 Stack Requirements3-2
	3.1.3	CL387 Register Usage 3-3
	3.1.4	80387 Control Word Settings3-4
		CL387 Numeric Exceptions3-4
	3.1.6	CL387 Real and Complex Functions3-8
3.2	CL38	7 Real Functions 3-9
	3.2.1	
		How to Read the Real Function Reference Pages3-12
		CL387 Real Function Reference
3.3	CL38	7 Complex Functions
	3.3.1	Summary of CL387 Complex Functions
	3.3.2	Complex Functions and Complex Numbers 3-92
	3.3.3	How to Read the Complex Function Reference Pages 3-94
	3.3.4	CL387 Complex Function Reference
Ch	apter	4 Exception Handling Library
4.1	Libra	ry Overview4-1
	4.1.1	Declaring EH387 Routines in ASM386 Exception Handlers 4-2
	4.1.2	Linking with EH3874-3
	4.1.3	EH387 Stack Requirements4-3
	4.1.4	EH387 Register Usage4-3
	4.1.5	EH387 Parameters and Results4-4
		ESTATE3874-5
		Protocols for Writing an EH387 Exception Handler4-10
	4.1.8	EH387 Exception Handler Template in ASM3864-11
4.2	EH38	7 Reference
	_	
Ap	pend	x A 80387 Support Library PUBLIC Symbols
A.1	Initia	alization Library PUBLIC SymbolsA-1
A.2		87 PUBLIC Symbols A-1
A.3	CL38	37 PUBLIC Symbols A-2
A.4	EH3	87 PUBLIC Symbols

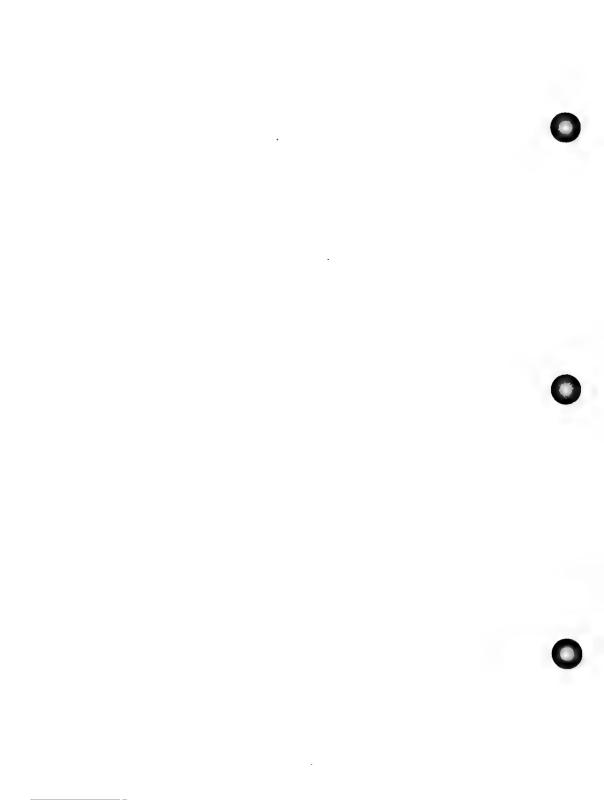
	Ap	pendix B High-level Languages and the 80387 Support Library
	B.1	Decimal Conversion Routines
	B.2	B.1.2 Far Call to mqcDECLOW_BIN
	B.3	B.2.2 Far Call to mqerIE2
	Apı	pendix C ANSI/IEEE Std 754-1985 Conformance
	C.1	Decimal Conversion Library
	C.2	Common Elementary Function Library
		Exception Handling Library
	App	pendix D 80387 Numeric Data Formats
	D.1	Integer FormatsD-1
	D.2	Real FormatsD-2
•	App	pendix E ASM386 Floating-Point Instructions
	App	pendix F 80387 Support Library Summary
	F.1	Parameters and Results for DC387 RoutinesF-1
	F.2	Arguments and Results for CL387 FunctionsF-3
	F.3	Parameters and Results for EH387 RoutinesF-6
		80387 Exceptions and Exception OpcodesF-8
	Glo	essary
	Ind	ex
	Serv	ice Information Inside Back Cover

# **Figures**

2-1	DCB Layout	2-4
2-2	ADCB Layout	
2-3	80387 Exception Byte for DC387	2-7
3-1	80387 Exception Byte for CL387	3-5
3-2	Rectangular Graph of a Complex Number	
3-3	Polar Graph of a Complex Number	3-94
3-4	Absolute Value of a Complex Number	3-96
3-5	Domain of Arccosh(z) with Branch Cuts	3-100
3-6	Range of Arccosh(z)	
3-7	Domain of Arccos(z) with Branch Cuts	3-104
3-8	Range of Arccos(z)	3-104
3-9	Domain of Arcsinh(z) with Branch Cuts	3-108
3-10	Range of Arcsinh(z)	3-108
3-11	Domain of Arcsin(z) with Branch Cuts	3-112
	Range of Arcsin(z)	
3-13	Domain of Arctanh(z) with Branch Cuts	3-116
3-14		
3-15	Domain of Arctan(z) with Branch Cuts	3-120
3-16	Range of Arctan(z)	3-120
3-17	Domain of Ln(z) with Branch Cut	3-155
3-18	Range of Ln(z)	3-155
3-19	Rectangular and Polar Forms of a Complex Number	3-162
3-20	Rectangular and Polar Forms of a Complex Number	3-171
4-1	ESTATE387 Layout	4-5
4-2	ARGUMENT Byte in ESTATE387	4-6
4-3	RESULT Byte in ESTATE387	4-8

# **Tables**

2-1	80387 State at DC387 Trap	2-9
	Decimal Conversion Library Routines	
2-3	mqcBIN_DECLOW Conversions of 80387 Special Values	2-13
3-1	80387 State at CL387 Trap	3-7
3-2	Common Elementary Real Functions	3-10
3-3	Common Elementary Complex Functions	3-90
4-1	Atyp and Rtyp Field Values	4-7
4-2	FORMAT Byte Values	4-9
C-1	Correctly Rounded Decimal Conversion Ranges Required	C-1
C-2	Decimal Conversion Ranges Required	
	Integer Formats	
D-2	Real Formats	D-2
D-3	Summary of Real Format Parameters	D-3
D-4	Real Zero, Infinity, and QNaN Indefinite Values	D-3
F-I	Summary of CL387 Real Function Arguments and Results	F-4
F-2	Summary of CL387 Complex Function Arguments and Results	F-5
F-3	Summary of Possible DC387 and CL387 Exceptions	F-8
	DC387 and CL387 Exception Opcodes	







This manual is a reference for the 80387 Support Library. It assumes that you are familiar with the 80387 numerics coprocessor for 80386-based systems. It has the following chapters and appendixes:

- Chapter 1, Numerics Support and 80387 Initialization, contains an overview of the 80387 Support Library and of its configurations and linkage requirements. It also contains an explanation of the 80387 Initialization Library.
- Chapter 2, Decimal Conversion Library, contains information about the decimal-to-binary, binary-to-decimal, and binary-tobinary conversion routines. Each routine has an ASM386 example.
- Chapter 3, Common Elementary Function Library, contains information about the algebraic, logarithmic, exponential, trigonometric, and hyperbolic real and complex functions. Each function has an ASM386 example.
- Chapter 4, Exception Handling Library, contains information about utility routines that make writing exception handlers for 80387 exceptions easier. This chapter has an ASM386 template for an exception handler, as well as an example for each routine.
- Appendix A, 80387 Support Library PUBLIC Symbols, lists the names of all PUBLIC symbols in the 80387 Support Library modules.
- Appendix B, High-level Languages and the 80387 Support
  Library, contains information about calling Support Library
  routines from high-level languages. This appendix has PL/M-386
  examples for representative routines.
- Appendix C, ANSI/IEEE Std 754-1985 Conformance, explains the relationship between the 80387 Support Library and the IEEE Standard for Binary Floating-Point Arithmetic.
- Appendix D, 80387 Numeric Data Formats, contains a reference for the 80387 integer and real formats.
- Appendix E, ASM386 Floating-Point Instructions, contains a summary of the operands, operation, and possible exceptions for each 80387 instruction. It also contains a list of exception opcodes returned by the exception handling utilities (see Chapter 4).

 Appendix F, 80387 Support Library Summary, contains a quick reference for the parameters, results, possible exceptions, and exception opcodes of the Decimal Conversion, Common Elementary Function, and Exception Handling routines.

A glossary and index follow the appendixes.

#### **Related Publications**

The following contain detailed information about 80386 architecture, the 80387 numerics coprocessor, and numerics processing:

- 80386 Programmer's Reference Manual, order number 230985
- 80386 System Software Writer's Guide, order number 231499
- 80387 Programmer's Reference Manual, order number 231917
- IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985)

The following contain information about linking and locating programs that call 80387 Support Library routines:

- Intel386<sup>™</sup> Family Utilities User's Guide, order number 481343
- Intel386<sup>™</sup> Family System Builder User's Guide, order number 481342

See also your language reference manual if you want to call 80387 Support Library routines in high-level language programs.

See the Release Notes for how to install the 80387 Support Library on your system.

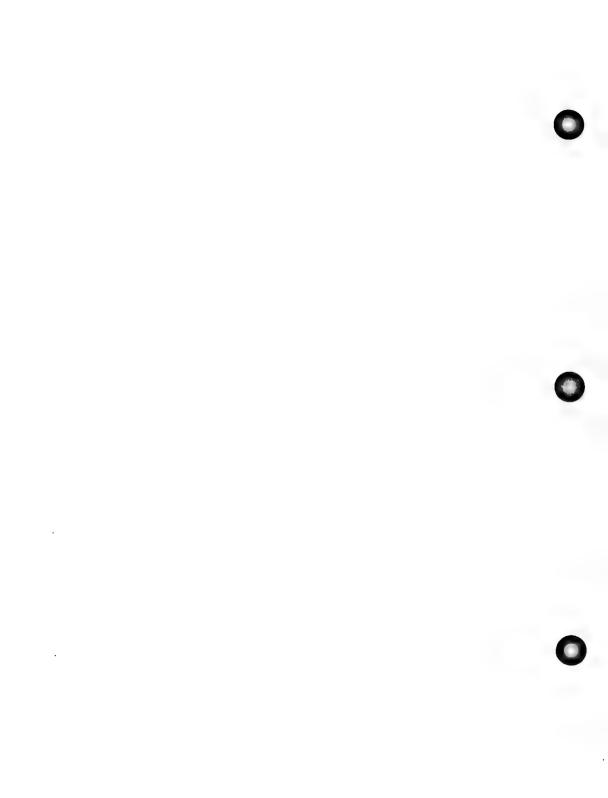




# **Contents**

# Chapter 1 Numerics Support and 80387 Initialization

1.1	80387 Support Library Overview1-	J
	1.1.1 80387 Support Library Configurations1-	
	1.1.2 80387 Support Library Linkage1-	
1.2	Initialization Library Overview1-	
	1.2.1 INIT87 and INITFP Routines	
	1.2.2 Declaring Initialization Routines in ASM386 Programs: 1-	



This chapter contains two major sections:

- 1. An overview of the 80387 Support Library
- 2. An explanation of the Initialization Library

# 1.1 80387 Support Library Overview

The 80387 Support Library is a collection of four functionally distinct libraries:

- 1. Initialization Library routines set up the numerics processing environment for 80386-based systems with an 80387 or true software emulator. See Section 1.2 for more information about the Initialization Library.
- 'Decimal Conversion Library (DC387) routines convert floatingpoint numbers from one 80387 binary storage format to another or from ASCII decimal strings to 80387 binary floating-point format and vice versa. See Chapter 2 for more information about DC387.
- 3. Common Elementary Function Library (CL387) routines perform algebraic, logarithmic, exponential, trigonometric, and hyperbolic operations on real and complex numbers, as well as real-to-integer conversions. See Chapter 3 for more information about CL387.
- 4. Exception Handling Library (EH387) routines make writing numerics exception handlers easier. See Chapter 4 for more information about EH387.

All 80387 Support Library modules are in 80386 Object Module Format (OMF386). They can be linked (see Section 1.1.2) with the OMF386 output of any Intel translator to execute on an 80387 or on an 80386 with a true software emulator. All routines are completely reentrant: working storage is on the 80387 stack, on the 80386 stack, or in the 80386 registers. All 80387 Support Library modules declare their routines as PUBLIC symbols (see Appendix A). You must declare each routine as an external symbol before it is called (see Section 1.2.2 and Appendix B for example declarations).

## 1.1.1 80387 Support Library Configurations

Each of the 80387 Support Libraries has near and far versions:

- The 80387N.LIB, DC387N.LIB, CL387N.LIB and EH387N.LIB modules contain the near versions of the 80387 Support Libraries:
  - Near library modules have a code segment named CODE32 and a single combined data/stack segment named DATA. The 80386 DS, ES, and SS registers are assumed to access the DATA segment.
  - Although parameters to the Decimal Conversion and Exception Handling routines are 48-bit pointers, only the low-order 32 bits of such parameters are meaningful within near library modules. Calls to library routines are offset only.
- The 80387F.LIB, DC387F.LIB, CL387F.LIB, and EH387F.LIB modules contain the far versions of the 80387 Support Libraries:
  - Far library modules have separate code, data, and stack segments.
  - Pointer parameters to the Decimal Conversion and Exception Handling routines are 48-bit segment:offset pairs, each pointing to a 16-bit segment base and 32-bit offset in 80386 memory. Calls to library routines are segment:offset.

80387 Support Library code is USE32: all offsets are 32 bits and the 80386 stack is 32 bits wide.

# 1.1.2 80387 Support Library Linkage

Any program with near or far calls to Support Library routines must be linked with the appropriate Support Library modules:

- CL387N.LIB or CL387F.LIB
- DC387N.LIB or DC387F.LIB
- EH387N.LIB or EH387F.LIB
- 80387N.LIB or 80387F.LIB

Link your code only to Support Library modules that contain routines your program calls. For example, link with CL387N.LIB and 80387N.LIB if your program calls only the INIT87 and mqerTAN routines in a code segment named CODE32 (see Section 1.1.1).

# 1.2 Initialization Library Overview

The 80386-based 80387 Initialization Library consists of two routines, INIT87 and INITFP, that set up the numerics processing environment for the following:

- Execution of floating-point instructions, whether in a high-level language (see Appendix B), in ASM386 (see Appendix E), or both
- Calls to 80387 Support Library routines in the Decimal Conversion Library (DC387), Common Elementary Function Library (CL387), and Exception Handling Library (EH387)

#### 1.2.1 INIT87 and INITFP Routines

INIT87 and INITFP initialize the 80387 and leave it in a known state, ready to perform floating-point operations. Both routines set up the following modes in the 80387 Control Word:

- Precision is extended (64-bit significand).
- Rounding is to nearest with even preferred (default 80387 rounding mode).

Both routines also clear 80387 exceptions and leave the 80387 stack empty on return to the caller. Neither routine requires parameters.

The only difference between INIT87 and INITFP is how each masks numeric exceptions in the 80387 Control Word:

- INIT87 masks all exceptions.
- INITFP masks all exceptions except I (Invalid Operation). Note that an 80387 stack fault is reported as an I exception.

After a call to INIT87 or INITFP, you can modify the Control Word to suit the needs of your program.

## 1.2.2 Declaring Initialization Routines in ASM386 Programs

The 80387 Initialization Library can be linked to code that is within the same code segment or to code that is in another code segment. However, you must declare the initialization routine within the calling module. For example, when a module contains a call to INIT87, make one of the following declarations:

• If you want your code to be within the same segment as the initialization, use the 80387N.LIB (near library). Declare INIT87 as follows before calling this routine:

CODE32 SEGMENT ER ; Segment name must be CODE32. ; Library segments are USE32. EXTRN INIT87: NEAR ; INIT87 is now callable. ; CODE32 ENDS

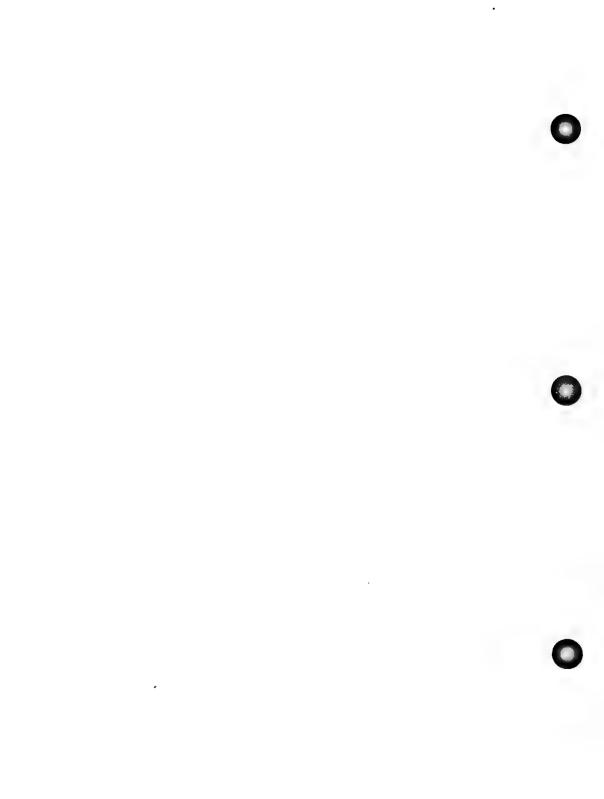
• If you want your code to be in a different segment from the initialization, use the 80387F.LIB (far library). Declare INIT87 as follows before calling this routine:

EXTRN INIT87: FAR ; Declaration must be outside ; all SEGMENT..ENDS pairs ; of the module. INIT87 is ; now callable within the ; module.





CI	apter 2 Decimal Conversion Library	
2.1	Library Overview	2-1
	2.1.1 Declaring DC387 Routines in ASM386 Programs	2-2
	2.1.2 Declaring DC387 Parameters and Data Structures	2-2
	2.1.3 DC387 Results	2-5
	2.1.4 DC387 Stack Requirements	
	2.1.5 DC387 Register Usage	2-6
	2.1.6 80387 Control Word Settings	
	2.1.7 DC387 Numeric Exceptions	
	2.1.7.1 DC387 Masked Exception Handling	
	2.1.7.2 DC387 Unmasked Exception Handling	
2.2	Decimal Conversion Library Routines	
	2.2.1 Summary of DC387 Routines	2-9
	2.2.2 How to Read the DC387 Reference Pages	
	2.2.3 DC387 Reference	
mad	CBIN DECLOW	
	cDEC BIN	
	CDECLOW BIN	
	CDUBL XTND	
	cSNGL XTND	
	CXTND_DUBL	
	cXTND_SNGL	
Fig	gures	
2-1	DCB Layout	2-4
	ADCB Layout	
	80387 Exception Byte for DC387	
	bles	
2-1	80387 State at DC387 Trap	2 (
	Decimal Conversion Library Routines	
	macRIN DECLOW Conversions of 20227 Special Values	



This chapter has two major sections:

- 1. An overview of the 80387 Decimal Conversion Library (DC387)
- 2. A reference for the DC387 routines

# 2.1 Library Overview

The 80386-based 80387 Decimal Conversion Library consists of routines that convert floating-point numbers either from one 80387 binary storage format to another or from ASCII decimal strings to 80387 binary floating-point format and vice versa. These routines can be linked with the OMF386 output of any Intel translator to execute on an 80387 or on an 80386 with a true software emulator.

For maximum execution speed, DC387 is coded in ASM386 assembly language. The DC387 decimal-to-binary, binary-to-decimal, and binary-to-binary conversion routines not only conform with the ANSI/IEEE754-1985 Standard for Binary Floating-Point Arithmetic, they exceed its requirements (see Appendix C for details).

The following subsections explain:

- How to declare DC387 routines in ASM386 programs
- How to declare the data structures for these routines
- How DC387 routines return results
- How DC387 uses the 80387 and 80386 stacks
- How DC387 uses the 80386 registers
- How the 80387 Control Word affects the DC387 routines and vice versa
- How DC387 detects, responds to, and reports 80387 exceptions

## 2.1.1 Declaring DC387 Routines in ASM386 Programs

The Decimal Conversion Library can be linked to code that is within the same code segment or to code that is in another code segment. However, you must declare each DC387 routine that your program calls within the calling module. For example, within a module that calls mqcXTND DUBL, make one of the following declarations:

 If you want your code to be within the same segment as DC387, use the DC387N.LIB (near library). Declare mqcXTND\_DUBL as follows before calling this routine:

CODE32 SEGMENT ER ; Segment name must be CODE32.

; DC387 segments are USE32.

EXTRN MQCXTND DUBL: NEAR

: ; MQCXTND\_DUBL is now callable.

• If you want your code to be in a different segment from DC387,

use the DC387F.LIB (far library). Declare mqcXTND\_DUBL as follows before calling this routine:

EXTRN MQCXTND\_DUBL: FAR ; Declaration must be

: outside all SEGMENT..ENDS

; pairs of the module.

Declare only those DC387 routines that each module calls and link to the appropriate DC387N.LIB or DC387F.LIB file before executing the program. Note that DC387 library modules use 32-bit addressing. See the reference pages in Section 2.2.3 for ASM386 examples of how to declare each DC387 routine. See Appendix B for more information about declaring DC387 routines in high-level language modules.

## 2.1.2 Declaring DC387 Parameters and Data Structures

Parameters to the DC387 routines are pointers pushed on the 80386 stack. The segment part of a far pointer must be pushed as a 4-byte quantity because DC387 uses a 32-bit wide stack. Any push of a segment register in a USE32 segment automatically pushes 4 bytes on the stack. ASM386 parameters to the DC387N.LIB routines must be declared in a combined data/stack segment named DATA.

DC387 routines that convert from one 80387 binary floating-point format to another require two parameters:

- A pointer to the input buffer where the value to be converted is stored in 80386 memory
- A pointer to the output buffer where the DC387 binary-to-binary routine stores its converted result

DC387 routines that convert from decimal to binary or from binary to decimal require one parameter: a pointer to a structured, contiguous block of storage in 80386 memory. The following declarations define the ASM386 structures for mqcDEC\_BIN, mqcDECLOW\_BIN, and mqcBIN\_DECLOW. Most ASM386 examples for these routines in this chapter assume that these declarations have been made.

```
: The following are named constants for both
: structures' PRSCN field. Values indicate an
: input/output real's format, using the same codes
; as the PRECISION field of the 80387 Control Word.
                          : for 32-bit single format
SNGL PRSCN EOU O
                          : for 64-bit double format
DUBL PRSCN EOU -8
                          : for 80-bit extended format
XTND PRSCN EQU &
: The following declaration is the Decimal Conversion
; Block (DCB) data structure. mgcDEC BIN's parameter
; points to a variable of this structure type.
DCB
      STRUC
  B BUF PTR DP ?
                          ; segment:offset (16:32 bits)
                          ; for binary output buffer
                          : DC387N.LIB uses offset only
                          ; (8-bit code) see preceding
  PRSCN
             DB ?
                          : constant definitions
                          ; (8-bit ordinal) length
 LGNTH
             DR ?
                          ; of input digit string
                          ; segment:offset (16:32 bits)
  D BUF PTR
                         ; for decimal input buffer
                          : DC387N.LIB uses offset only
DCB
      ENDS
```

Figure 2-1 illustrates the DCB data structure, followed by the declaration of the ADCB data structure

	Byte		e off	
	B_BUF_PTR	OFFSET		
LNGTH	PRSCN	B_BUF_PTR	SEGMENT*	
	D_BUF_PTR	OFFSET		
·		D_BUF_PTR	SEGMENT*	7 1

\*Meaningful only for DC387F.L!B

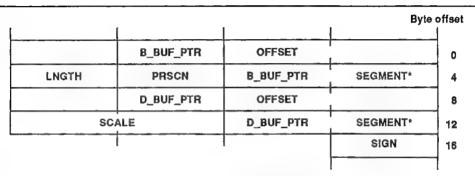
Figure 2-1 DCB Layout

PCD0001

```
; The following declaration defines the Alternative ; Decimal Conversion Block for mqcDECLOW_BIN (decimal ; to binary) and the Augmented Decimal Conversion ; Block (ADCB) for mqcBIN_DECLOW (binary to decimal). ; mqcDECLOW_BIN's and mqcBIN_DECLOW's parameters point ; to variables of the ADCB structure type.
```

```
ADCB
       STRUC
                           : initial ADCB fields accessed
                           ; by DCB field names:
           DP ?
                           ; B BUF PTR to binary buffer:
                           ; output for Alternative block
                           ; or input for Augmented block
           DB ?
                           ; PRSCN: format, Alternative
                           : output or Augmented input
                           : LGNTH: string, Alternative
           DB ?
                           ; output or Augmented input
                           : D BUF PTR to decimal buffer:
           DP ?
                           : input for Alternate block or
                           ; output for Augmented block
  SCALE
           DW ?
                           : (16-bit) two's complement
                           ; integer is value of true
                           ; decimal exponent
                           : (8-bit) ASCII value
  SIGN
           DB ?
ADCB
      ENDS
```

Figure 2-2 illustrates the ADCB data structure.



\*Meaningful only for DC387F,LIB

Figure 2-2 ADCB Layout

PCD0002

#### 2.1.3 DC387 Results

The DC387 routines are external procedures that produce results in two ways:

- Each DC387 routine uses one or two pointer parameters to access floating-point values in 80386 memory. Each converts an input value from one representation to another, stores this result in another 80386 memory location, and pops all input parameters from the 80386 stack before returning control to its caller.
- 2. Every DC387 routine returns an updated copy of the 80387 exception byte (see Section 2.1.7) to the AL register before returning control to its caller.

As a language, ASM386 does not distinguish between routines that are procedures (produce results indirectly) and those that are functions (return typed results directly). However, many high-level languages do distinguish between procedures and functions. Note that a DC387 routine might be either a procedure or a function in such a language, depending on how it is declared. See Appendix B for more information about declaring DC387 routines in high-level languages.

#### 2.1.4 DC387 Stack Requirements

DC387 routines save the entire 80387 state, including its stack, when they are called. They restore the 80387 to its original state before returning control to the caller. Therefore, DC387 does not require you to leave 80387 stack positions free for its use.

All DC387 routines are completely reentrant: they use no fixed 80386 memory locations to store internal variables. However, any procedure that interrupts DC387 should save the entire 80387 stack upon entry and restore it before returning control to DC387.

DC387N.LIB requires 240 bytes on the 80386 stack for its internal storage; DC387F.LIB requires 288 bytes. DC387 itself allocates the required stack bytes within its modules. However, a program such as a numeric exception handler could interrupt DC387 and then call DC387 again. If this occurs, the exception handler should allocate an additional 240 (near library) or 288 (far library) bytes on the 80386 stack at each recursion.

## 2.1.5 DC387 Register Usage

DC387 conforms to the register-saving conventions of Intel 80386 high-level languages. According to these conventions:

- The DC387 routines must leave the 80386 DS, ES, SS, and EBP registers unchanged. For DC387N.LIB, ES is assumed to access the same combined data/stack segment (DATA) as DS.
- The DC387 routines may destroy the contents of the EAX, EBX, ECX, EDX, EDI, ESI, FS, and GS registers.

## 2.1.6 80387 Control Word Settings

All DC387 routines save the 80387 Control Word when they are called and restore it before returning control to the caller. However, DC387 routines sometimes use a different rounding mode than their callers:

 The binary-binary routines mqcXTND\_DUBL, mqcXTND\_SNGL, mqcDUBL\_XTND, and mqcSNGL\_XTND round results according to the caller's rounding mode.  The decimal-binary routines mqcDEC\_BIN, mqcDECLOW\_BIN, and mqcBIN\_DECLOW always round results to nearest with even preferred, regardless of the caller's rounding mode.

DC387 places no constraints on the setting of exception masks in the 80387 Control Word (see Sections 2.1.7.1 and 2.1.7.2).

#### 2.1.7 DC387 Numeric Exceptions

DC387 routines signal some of the same numeric exceptions as the 80387 instructions. DC387 routines report 80387 exceptions by setting appropriate bits in the 80387 exception byte, which is the low-order byte of the Status Word. The DC387 routines copy this byte to the AL register (see Section 2.1.3) before restoring the caller's 80387 state and returning control to the caller.

Figure 2-3 illustrates the 80387 exception byte, with possible DC387 exceptions shown in bold-faced type (S and Z never occur).

• •	<b>3</b> 2		,
P U	0 Z	ו מ	
	T. I.		P U O Z D I

The DC387 routines behave like atomic instructions in detecting, responding to, and reporting the I, D, O, U, and P numeric exceptions:

- I DC387 signals the Invalid Operation exception for such programming errors as:
  - Input floating-point values in 80387 unsupported formats (pseudozeros, psuedo-NaNs, pseudoinfinities, and unnormals)
  - Input floating-point values that cannot be converted as specified by the combination of ADCB field values (see mqcBIN\_DECLOW)
- D DC387 signals the Denormal exception for an input denormal value.

- O DC387 signals the numeric Overflow exception when the exponent of a rounded result is too large for the format of the destination.
- U DC387 signals the numeric Underflow exception when a result is too tiny to be represented accurately in the destination format if underflow is masked. It signals the Underflow exception when a normalized result is too tiny for the destination format if underflow is unmasked.
- P DC387 signals the numeric Precision (inexact) exception when a result must be rounded to fit in the destination format.
- E DC387 sets the Exception Status bit (7) if it sets any of the I, D, O, U, and P bits (0, 1, 3..5) of the 80387 exception byte (see Figure 2-3).

DC387 routines are like atomic instructions: each routine reports exceptions when the mathematical definition of the routine requires it to take exception to particular input values or results.

## 2.1.7.1 DC387 Masked Exception Handling

DC387 responds to masked exceptions just as the 80387 does. It sets the appropriate bit of the 80387 exception byte and supplies a default result. See each routine in Section 2.2.3 for specific information about default results for masked exceptions.

# 2.1.7.2 DC387 Unmasked Exception Handling

DC387 provides exception handling information in the same data structure that the 80387 uses for handling unmasked exceptions: the 80387 State.

When an unmasked exception generates an Interrupt 16 (Processor Extension Error) from a DC387 routine, the interrupt comes from a DC387 code site where cleanup of the 80386 stack is about to occur just before DC387's return to its caller. When an exception handler for the Processor Extension Error executes its IRETD, the return is to this cleanup section. The DC387 routine first restores the caller's 80387 State and then returns control to its caller.

At the time of such a trap from DC387, the 80387 State consists of the 80387 Environment and 80387 stack information summarized in Table 2-1.

Table 2-1 80387 State at DC387 Trap

Field	Contains
Control Word	caller's Control Word
Status Word	(caller's Status Word OR DC387 exceptions)
Tag Word	80387 stack, as shown (see STST(7) following)
EIP Offset	FFFFFFFH for USE32 protected mode
or Instruction Pointer	FFFFFFFH for USE32 real address mode
Opcode	DC387 routine's exception opcode
ST	DC387 routine's input or result
ST(1)ST(7)	empty

An exception handler for the Processor Extension Error can examine the 80387 Instruction Pointer or EIP Offset value to determine whether DC387 generated an Interrupt 16. It can identify which DC387 routine caused the interrupt by examining the Opcode field value (see Table 2-1).

## 2.2 Decimal Conversion Library Routines

This section contains:

- A summary of the DC387 routines
- Information about how to read each routine's reference pages
- A comprehensive reference for each DC387 routine in alphabetic order

## 2.2.1 Summary of DC387 Routines

Table 2-2 lists the DC387 routines, their pointer parameters, and what each routine does. See Section 2.1.2 for a general discussion of the parameters to these routines. See each routine in Section 2.2.3 for details.

Table 2-2 Decimal Conversion Library Routines

Binary to Binary Rou	ıtines		
Name	Parameters	Description	
mqcXTND_DUBL	XTND_PTR	Converts binary value in 80-bit	
_	DUBL_PTR	extended format to same value	
		in 64-bit double format	
mqcXTND_SNGL	XTND_PTR	Converts binary value in 80-bit	
	SNGL_PTR	extended format to same value	
		in 32-bit single format	
mqcDUBL_XTND	DUBL_PTR	Converts binary value in 64-bit	
	XTND_PTR	double format to same value	
A1101 10010	A1101	in 80-bit extended format	
mqcSNGL_XTND	SNGL_PTR	Converts binary value in 32-bit	
	XTND_PTR	single format to same value	
		in 80-bit extended format	
Decimal to Binary Ro	outines		
Name	Parameter	Description	
mqcDEC_BIN	DCB_PTR	Converts decimal string	
		to 80387 binary format	
mqcDECLOW_BIN	ADCB_PTR	Converts decimal string with low-	
		level interface to binary format	
Binary to Decimal Ro	outine		
Name	Parameter	Description	
mqcBIN_DECLOW	ADCB_PTR	Converts 80387 binary format	
	_	value to decimal string	
		with low-level interface	

Every DC387 routine begins with the prefix mqc. This prefix reduces the chance of conflict between a DC387 name and another name in your program. DC387 also has alternate PUBLIC names beginning with mqc for its routines. These names, listed in Appendix A, are used by some Intel translators.

To make the DC387 routines' names more readable in this chapter, the mqc prefix is in lowercase letters, except in examples. However, uppercase and lowercase names are interchangable within DC387 library modules.

## 2.2.2 How to Read the DC387 Reference Pages

The reference pages for the DC387 routines have five sections:

- 1. Parameter describes the pointer argument to a decimal-binary conversion routine, along with the DCB- or ADCB-type (see Section 2.1.2) data structure it accesses. This section also describes the values allowed for each field of the structure.
- or Parameters describe the pointer arguments to a binary-binary conversion routine, along with the floating-point data each pointer accesses.
- Discussion explains what the routine does, whether there are any additional requirements on the input data, and how the routine handles special input values or what results it returns for certain inputs, or both.
- 3. Exceptions first summarizes which 80387 exceptions the routine can report. Then, it explains what causes each exception and how the routine handles the masked and unmasked cases.
- 4. Exception Opcode has the hexadecimal value the routine stores in the Opcode field of the 80387 State if the routine generates a trap (see Table 2-1).
- 5. Example has a commented ASM386 example that shows how to declare the routine NEAR or FAR, how to declare its pointer argument(s) and data, and how (in what sequence) to push its argument(s) on the 80386 stack before calling the routine.

## 2.2.3 DC387 Reference

The remaining pages in this chapter are a comprehensive reference for each Decimal Conversion Library routine. The routines are in alphabetic order.

## mgcBIN DECLOW

Converts binary number to decimal string with low-level interface

#### **Parameter**

ADCB\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to an Augmented Decimal Conversion Block (see ADCB in Section 2.1.2) in 80386 memory. The 17-byte ADCB has six fields containing the following variables:

- B\_BUF\_PTR is a 48-bit pointer to the binary input buffer. For DC387N.LIB, only the low-order 32 bits of B\_BUF\_PTR are meaningful. The input buffer contains a 32-bit single, 64-bit double, or 80-bit extended format value.
- PRSCN is a one-byte input code that specifies whether
   B\_BUF\_PTR points to a single, double, or extended format
   binary number. Valid PRSCN codes are: 0 for single, 2 for
   double, and 3 for extended format. B\_BUF\_PTR should point to
   a number whose format matches the PRSCN code. Otherwise,
   mqcBIN\_DECLOW returns undefined results to D\_BUF\_PTR,
   SCALE, and SIGN.
- LNGTH is a one-byte input ordinal from 0 to 255, inclusive, that specifies how many decimal digits mqcBIN\_DECLOW will output. Specify at least 9 for single format or 17 for double format to always obtain discrete decimal strings from adjacent binary input values. mqcBIN\_DECLOW returns only the SCALE and SIGN values if LNGTH is zero and B\_BUF\_PTR does not point to a negative zero, an infinity, or a NaN.
- D\_BUF\_PTR is a 48-bit pointer to the decimal output buffer.
  For DC387N.LIB, only the low-order 32 bits of D\_BUF\_PTR are
  meaningful. mqcBIN\_DECLOW returns a string of ASCII decimal
  digits or special values (see Table 2-3) to the output buffer. It
  does not return an explicit decimal point.
- SCALE is an output value, a 16-bit integer in two's complement.
   In SCALE, mqcBIN\_DECLOW returns the value of the true base<sub>10</sub> exponent for the digits at D\_BUF\_PTR.
- SIGN is a byte output in ASCII. mqcBIN\_DECLOW returns +
   (2BH) for a positive input, (2DH) for a negative input, and .
   (2EH) for a NaN input.

## mqcBIN\_DECLOW (continued)

#### Discussion

mqcBIN\_DECLOW converts the binary value at B\_BUF\_PTR into a decimal representation stored at D\_BUF\_PTR, SCALE, and SIGN. mqcBIN\_DECLOW's caller can use these values to construct the final output format of the number. Table 2-3 shows what mqcBIN\_DECLOW returns for input 80387 special values.

Table 2-3 mqcBIN\_DECLOW Conversions of 80387 Special Values

Input Binary	SIGN in	SCALE in Two's	Decimal Output Buffer in ASCII
Value	(hex)	Complement	(hex)
NaN	. (2EH)	32767	. (2EH) followed by blanks (20H)
+∞	+ (2BH)	32767	+ (2BH) followed by blanks (20H)
	- (2DH)	32767	- (2DH) followed by blanks (20H)
+0	0 (30H)	0	all blanks (20H)
-0	- (2DH)	0	0 (30H) followed by blanks (20H)

mqcBIN\_DECLOW returns only digits at D\_BUF\_PTR; an implicit decimal point exists after the rightmost digit. If LNGTH is greater than 18, mqcBIN\_DECLOW fills the decimal output buffer with 18 decimal digits, followed by ASCII underscore (5FH) characters with an assumed decimal point after the rightmost underscore.

## **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

Input -0, ±∞, or NaN at B\_BUF\_PTR when the LNGTH field's value is 0 causes the I exception. If I is masked, mqerBIN\_DECLOW returns appropriate SIGN and SCALE results (see Table 2-3), but it does not return digits at D\_BUF\_PTR. If I is unmasked, control passes to the exception handler with the input value in ST (the 80387 stack top) and the exception opcode set.

## mqcBIN DECLOW (continued)

- D Input denormals at B\_BUF\_PTR cause the D exception. If D is masked and the input is in single or double format, mqcBIN\_DECLOW returns correct results as if for a normalized value in extended format. If D is masked and the input is in extended format, it returns results as if for a value with the same significand but with a biased exponent of +1 (minimum true exponent -16382). If D is unmasked, control passes to the exception handler with the input number in ST and the exception opcode set.
- P A result that must be rounded to fit in the destination format causes the P exception. If P is masked, mqcBIN\_DECLOW returns the rounded result. If P is unmasked, control passes to the exception handler with the input number in ST and the exception opcode set.

## **Exception Opcode**

DIH

## Example

This example assumes the declarations of the ADCB structure and named PRSCN constants as in Section 2.1.2. It does not assume the declaration of mqcBIN DECLOW.

```
; DATA segment declared here.
                          ; binary input value (const)
REAL VAR3 DD OCOE9A36DR
                          ; for decimal output
D BUF3 DP ?
FINAL BUF DB 15 DUP (?)
BLOCK\overline{3} ADCB < ?. SNGL PRSCN.9.?.? >
             : BLOCK3 fields initialized to:
                          : B BUF PTR: undefined
                          : PRSCN: 32-bit single format
                          : LNGTH: 9 digits
                          ; D BUF PTR: undefined
                          ; SCALE: undefined (output)
                          ; SIGN: undefined (output)
                          ; CODE32 segment declared.
EXTRN MQCBIN DECLOW: NEAR
; Assume SS and DS already point to DATA (combined
; data/stack segment) for linkage with DC387N.LIB.
: The following lines initialize PTR fields, set up
; access to BLOCK3, and call MOCBIN DECLOW.
LEA EDX, REAL VAR3
                          ; EDX := offset(REAL VAR3)
MOV DWORD PTR BLOCK3.B BUF PTR, EDX
LEA EDX, D BUF3
                          : EDX := offset(D BUF3)
MOV DWORD PTR BLOCK3.D BUF PTR, EDX
LEA EDX, BLOCK3
                          : EDX := offset(BLOCK3)
PUSH EDX
                          ; BLOCK3 access onto
                          : 80386 stack
CALL MQCBIN DECLOW
                          ; Convert binary to decimal.
; DIGITS BUFFER is now 373330313139393434H (ASCII for
; string 730119944), SIGN is 2DH (ASCII for -), and
; SCALE is OFF8H (-8). Use these values to construct
; a number in any desired format at FINAL BUF.
```

## macDEC BIN

Converts decimal string to binary number

#### **Parameter**

DCB\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to a Decimal Conversion Block (see DCB in Section 2.1.2) in 80386 memory. The 14-byte DCB has four fields containing the following variables:

- B\_BUF\_PTR is a 48-bit pointer to the output binary number.
   For DC387N.LIB, only the low-order 32 bits of B\_BUF\_PTR are meaningful. The output buffer will contain a 32-bit single, 64-bit double, or 80-bit extended format value.
- PRSCN is a one-byte code that specifies whether B\_BUF\_PTR points to a single, double, or extended format binary number.
   Valid PRSCN codes are: 0 for single, 2 for double, and 3 for extended format. mqcDEC\_BIN returns undefined output if PRSCN has any other value.
- LNGTH is a one-byte input ordinal from 1 to 255, inclusive, that specifies the number of characters to be read at D\_BUF\_PTR. If LNGTH is 0, mqcDEC BIN returns undefined output.
- D\_BUF\_PTR is a 48-bit pointer to the input string of ASCII decimal characters. For DC387N.LIB, only the low-order 32 bits of D\_BUF\_PTR are meaningful. The caller should verify correct input syntax (see the following Discussion) before calling mqcDEC\_BIN; it returns undefined output when given input with invalid syntax.

## Discussion

mqcDEC\_BIN converts the string at D\_BUF\_PTR into a binary floating-point result in the format specified by PRSCN. It stores the binary value in 80386 memory at B\_BUF\_PTR.

mqcDEC\_BIN decomposes the input string into a significand with at most 21 digits and an implicit decimal point after the rightmost nonzero digit, an adjusted exponent, and a sign. Then it calls mqcDECLOW\_BIN.

## mqcDEC BIN (continued)

The ASCII string input to mqcDEC\_BIN must conform to the following rules:

- The string may have trailing blank characters (20H) but not leading or internal blanks.
- The initial character must be a (2DH) for a negative input value. Unsigned strings are positive, so a + (2BH) is optional.
- The immediately following characters are the significand digits:
  - o There must be at least one digit (30H = 0 through 39H = 9) in the significand string. The significand may have leading or trailing zeros (30H), but mqcDEC\_BIN discards them and passes the remaining digits to mqcDECLOW\_BIN if there are no more than 20 digits. If more than 20 digits remain, mqcDEC\_BIN collects the 20 most significant digits, appends the least significant digit to this string, and calls mqcDECLOW\_BIN.
  - The decimal point (2EH) is an optional character in the significand string. It may be the initial, the final, or an internal character of the string.
- An exponent string following the significand is optional:
  - The exponent's initial character must be an E (45H or 65H), a
     D (44H or 64H), or a T (54H or 74H). For DC387, these uppercase and lowercase letters are interchangable.
  - The next character must be a (2DH) for a negative exponent. An unsigned string is positive, so a + (2BH) is optional.
  - The remaining characters must be a string of at least one digit.
- The entire input string has an upper limit of 255 characters. A
  valid string must have at least one significand digit; the minimum
  length input is one byte containing a single ASCII digit. For
  example:
  - o 100.00 1E2 1T2 1.E2 .1E3 +100000d-3 are all valid input strings for the value 100, provided that there are no leading blanks (20H) in any string.
  - E2 1 E2 1.d .e2 -34.25 T-009
     are invalid input strings because the significand is missing (E2, .e2), an exponent digit is missing (1.d), or there are internal blanks (1 E2, -34.25 T-009). Leading blanks also make an input string invalid.

# mqcDEC\_BIN (continued) Exceptions

Possible exceptions are Overflow, Underflow, and Precision:

- O A result too large to fit in the destination format causes the O exception. If O is masked, mqcDEC\_BIN returns a correctly signed infinity at B\_BUF\_PTR. If O is unmasked but the result will fit in extended format, control passes to the exception handler with the output—its significand rounded to the format specified for PRSCN—in ST (the 80387 stack top) and the exception opcode set. If O is unmasked and the result is too large to fit in extended format, control passes to the exception handler with the QNaN indefinite in ST and the exception opcode set.
- U A result too tiny to fit accurately in the destination format causes the U exception if U is masked. In this case, mqcDEC\_BIN returns a gradual underflow denormal at B\_BUF\_PTR. A normalized result too tiny to fit in the destination format causes the U exception if U is unmasked. If U is unmasked but the result can be represented in extended format, control passes to the exception handler with the output—its significand rounded to the format specified for PRSCN—in ST and the exception opcode set. If U is unmasked and the result cannot be represented in extended format, control passes to the exception handler with the QNaN indefinite in ST and the exception opcode set.
- P A result that must be rounded to fit in the destination format causes the P exception. If P is masked, mqcDEC\_BIN returns the rounded result. If P is unmasked, control passes to the exception handler with an extended format result in ST and the exception opcode set.

## **Exception Opcode**

D<sub>0</sub>H

## Example

This example assumes the declarations of the DCB structure and named PRSCN constants as in Section 2.1.2. Exceptions are masked in the 80387 Control Word (see Section 2.1.6), so mqcDEC\_BIN returns default results to REAL\_VAR if exceptions occur; it also returns the 80387 exception byte to the AL register (see Sections 2.1.3 and 2.1.7). However, this example does not assume the declaration of mqcDEC\_BIN.

```
XCPTNS
         DR 0
                          : for 80387 exception byte
                          : on return from MOCDEC BIN
D IN BUF DB 100 DUP (?)
                          : decimal input buffer
ACTUAL
         DW 15
                          : for length of string
                          : binary output access
REAL VAR DO 0
BLOCK1 DCB < ?, DUBL PRCSN, ACTUAL, ?>
             : BLOCK1 fields initialized to:
                          : B BUF PTR: undefined
                          : PRSCN: 64-bit double format
                          : LNGTH: 15 (at assembly)
                          ; D BUF PTR: undefined
EXTRN MQCDEC BIN: NEAR
                          : in CODE32 segment
FNTRY:
: Assume that the decimal string -3.4E2 has already
; been moved into D IN BUF. The following lines
; initialize BUF PTR fields, set up access to BLOCK1,
; and call MQCDEC BIN in the CODE32 segment.
LEA EBX. REAL VAR
                          : EBX := offset(REAL VAR)
MOV DWORD PTR BLOCK1.B BUF PTR, EBX
                          EBX := offset(D IN BUF)
LEA EBX. D IN BUF
MOV DWORD PTR BLOCK1.D BUF PTR, EBX
LEA EBX. BLOCK1
                          ; EBX := offset(BLOCK1)
                          : BLOCK1 access onto
PUSH EBX
                          : 80386 stack
CALL MOCDEC BIN
                          : Convert decimal to binary.
                          : Store 80387 exception byte
MOV XCPTNS, AL
                          : in 80386 memory.
: REAL VAR is now the 64-bit double format
; representation of -340. If the value in REAL VAR
; is suspect, examine XCPTNS to see whether an \overline{8}0387
; exception bit was set during MQCDEC BIN.
```

## mqcDECLOW BIN

Converts decimal string with low-level interface to binary number

#### **Parameter**

ADCB\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to an Alternative Decimal Conversion Block (see ADCB in Section 2.1.2) in 80386 memory. The 17-byte ADCB has six fields containing the following variables:

- B\_BUF\_PTR is a 48-bit pointer to the output binary number.
   For DC387N.LIB, only the low-order 32 bits of B\_BUF\_PTR are meaningful. The output buffer will contain a 32-bit single, 64-bit double, or 80-bit extended format value.
- PRSCN is a one-byte code that specifies whether B\_BUF\_PTR points to a single, double, or extended format binary number.
   Valid PRSCN codes are: 0 for single, 2 for double, and 3 for extended format. mqcDECLOW\_BIN returns undefined output if PRSCN has any other value.
- LNGTH is a one-byte input ordinal from 1 to 21, inclusive, that specifies the number of significand digits to be read at D\_BUF\_PTR. It is mathematically impossible to map certain adjacent input strings to discrete binary values when both strings are longer than 6 digits for single, 15 for double, or 18 for extended format. If LNGTH less than 1 or greater than 21, mqcDECLOW\_BIN returns undefined output.
- D\_BUF\_PTR is a 48-bit pointer to the input ASCII string of decimal digits (30H through 39H). For DC387N.LIB, only the low-order 32 bits of D\_BUF\_PTR are meaningful. The input digits represent the significand as an integer without leading or trailing zeros and with an implicit decimal point after the rightmost digit. Adjust the exponent value to place the implicit decimal point correctly, store the adjusted exponent in SCALE, and store the sign in SIGN before calling mqcDECLOW BIN.
- SCALE is a 16-bit integer, expressed in two's complement. The SCALE value specifies the true base<sub>10</sub> exponent for the significand digits at D\_BUF\_PTR.
- SIGN is an input byte value in ASCII, either + (2BH) or (2DH).

#### Discussion

mqcDECLOW\_BIN accepts a pointer to a decimal number that has already been converted from some other format to the low-level format described under D\_BUF\_PTR, SCALE, and SIGN in the Parameter section. It converts the number into a binary floating-point number in the format specified by PRSCN and stores its result at B\_BUF\_PTR. mqcDECLOW\_BIN converts at most 20 digits in the significand string; it interprets a 21-digit string as a value that must be rounded. The twenty-first digit, if any, must be the least significant nonzero digit of a significand with more than 20 digits.

## **Exceptions**

Possible exceptions are Overflow, Underflow, and Precision:

- O A result too large to fit in the destination format causes the O exception. If O is masked, mqcDECLOW\_BIN returns a correctly signed infinity at B\_BUF\_PTR. If O is unmasked but the result will fit in extended format, control passes to the exception handler with the output—its significand rounded to the format specified for PRSCN—in ST (the 80387 stack top) and the exception opcode set. If O is unmasked and the result is too large to fit in extended format, control passes to the exception handler with the QNaN indefinite in ST and the exception opcode set.
- U A result too tiny to fit accurately in the destination format causes the U exception if U is masked. In this case, mqcDECLOW\_BIN returns a gradual underflow denormal at B\_BUF\_PTR. A normalized result too tiny to fit in the destination format causes the U exception if U is unmasked. If U is unmasked but the result can be represented in extended format, control passes to the exception handler with the output—its significand rounded to the format specified for PRSCN—in ST and the exception opcode set. If U is unmasked and the result cannot be represented in extended format, control passes to the exception handler with the QNaN indefinite in ST and the exception opcode set.
- P A result that must be rounded to fit in the destination format causes the P exception. If P is masked, mqcDECLOW\_BIN returns the rounded result. If P is unmasked, control passes to the exception handler with an extended format result in ST and the exception opcode set.

# mqcDECLOW\_BIN (continued) Exception Opcode

D<sub>0</sub>H

## Example

This example assumes definition of the named PRSCN constants as in Section 2.1.2. It does not assume the definition of the ADCB structure or of mqcDECLOW\_BIN.

```
EXTRN MOCDECLOW BIN: FAR
             : declared outside all SEGMENT. ENDS
             : in module
DECOMPOSED DEC INPUT SEGMENT RW USE32
  EXP1 DW -2
  SIGN1 DB '+'
  SIG DIGITS1 DB 15 : for length of string
  D BUF1 DB '371852946173258'
DECOMPOSED DEC INPUT ENDS
BINARY OUTPUT SEGMENT RW USE32
  DUBL REAL1 DO ?
                    : binary output
BINARY OUTPUT ENDS
CONVERT TO DUBL SEGMENT ER USE32
O XCPTN EQU 00001000B ; for exception masks
U_XCPTN EQU 00010000B ; of mqcDECLOW BIN's P XCPTN EQU 00100000B ; possible exceptions
                          ; possible exceptions
: DUBL FROM DCOMP routine can be called from another
; module where it has been declared EXTRN. It saves
; data segment registers, sets up a stack frame for
; ADCB data from 2 segments, and calls MQCDECLOW BIN.
; DUBL FROM DCOMP also checks for exceptions that
; occurred during MQCDECLOW BIN's conversion.
  DUBL FROM DCOMP PROC FAR
    PUSH DS
                          : 4-byte pushes of DS and ES
                  ; in USE32 segments
    PUSH ES
    PUSH EBP
    MOV EBP_ESP
```

## mqcDECLOW BIN (continued)

```
PUSH AX; Dummy push aligns stack on 4-bvte
             : boundary before call to MQCDECLOW BIN.
    MOV AX, DECOMPOSED DEC INPUT
    MOV DS, AX
  ASSUME DS: DECOMPOSED DEC INPUT
             ; Begin setup of ADCB on stack.
    PUSH DWORD PTR EXP1
                         : AX=16-bit segment(D BUF PTR)
    PUSH AX
    LEA ESI, D_BUF1 ; ESI := offset(D_BUF_PTR)
    PUSH ESI
    MOV AH, SIG DIGITS1
    MOV AL. DUBL PRSCN
    PUSH AX
                         : LNGTH and PRSCN onto stack
    MOV AX, BINARY OUTPUT
    PUSH AX
                         : AX=16-bit segment(B BUF PTR)
    MOV ES. AX
  ASSUME ES: BINARY OUTPUT
    LEA EDI DUBL REAL1 ; EDI := offset(B BUF PTR)
    PUSH EDI
             : Now ADCB data structure is on stack.
    MOV EAX, ESP
                        ; segment(ADCB PTR)
    PUSH SS
   PUSH EAX
                         : offset(ADCB PTR)
    CALL MOCDECLOW BIN
: also returns 80387 exception byte to AL register
   TEST AL, O XCPTN + U XCPTN + P XCPTN
   JNZ XCPTN REVIEW
            ; DUBL FROM DCOMP
EXIT:
   MOV ESP, EBP; MQCDECLOW BIN preserves EBP.
    POP ES
    POP DS
    RET
XCPTN REVIEW:
            ; Code to make note of exceptions that
             ; occurred in conversion goes here.
JMP EXIT
DUBL FROM DCOMP
                ENDP
CONVERT TO DUBL ENDS
```

## mgcDUBL XTND

Converts number in double format to same value in extended format

#### **Parameters**

DUBL\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to the input binary number in 80386 memory. If the input is not in 64-bit double format, mqcDUBL\_XTND returns undefined output at XTND\_PTR.

XTND\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to the binary output buffer.

#### Discussion

mqcDUBL\_XTND converts a binary floating-point value in 64-bit double format to the same value in 80-bit extended format.

## **Exception**

D Input denormals cause the D exception. If D is masked, mqcDUBL\_XTND returns the input value—normalized in extended format—at XTND\_PTR. If D is unmasked, control passes to the exception handler with the input—normalized in extended format—in ST (the 80387 stack top) and the exception opcode set.

## **Exception Opcode**

D4H

## mqcDUBL XTND (continued)

## Example

This example assumes that mqcDUBL\_XTND has not been declared.

```
EXTRN MOCDUBL XTND: FAR
             : declared outside all SEGMENT..ENDS
             : in module
             : Module's stack allocated and
             : data segments declared here.
DUBL NUM DQ ?
                          : 64-bit input
XTND NUM DT ?
                         : 80-bit output
            : SOMECODE USE32 segment declared here.
: Assume SS points to stack and DS contains the 16-bit
: segment part of far pointers to DUBL NUM
; and XTND NUM. The following lines convert the
: double format number at DUBL NUM to an extended
: format number stored at XTND NUM.
PUSH DS
                          : segment(DUBL NUM) onto 80386
                          ; stack (4-byte push)
LEA EDX, DUBL NUM
                          : EDX := offset(DUBL NUM)
                         : DUBL NUM offset onto stack
PUSH FDX
                          ; segment(XTND NUM) onto stack
PUSH DS
                         : EDX := offset(XTND NUM)
LEA EDX, XTND NUM
                          : XTND NUM offset onto stack
PUSH EDX
CALL MOCDUBL XTND
```

; XTND\_NUM now contains the same value as DUBL\_NUM.

## mqcSNGL XTND

Converts number in single format to same value in extended format

#### **Parameters**

SNGL\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to the input binary number in 80386 memory. If the input is not in 32-bit single format, mqcSNGL\_XTND returns undefined output at XTND\_PTR.

XTND\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to the binary output buffer.

#### Discussion

mqcSNGL\_XTND converts a binary floating-point value in 32-bit single format to the same value in 80-bit extended format.

## Exception

D Input denormals cause the D exception. If D is masked, mqcSNGL\_XTND returns the input value—normalized in extended format—at XTND\_PTR. If D is unmasked, control passes to the exception handler with the input—normalized in extended format—in ST (the 80387 stack top) and the exception opcode set.

## **Exception Opcode**

D2H

## mqcSNGL XTND (continued)

## Example

This example assumes that mqcSNGL\_XTND has not been declared.

```
: DATA segment declared here.
                         : 32-bit input
SNGL X
       DD ?
XTND X DT ?
                          : 80-bit output
                          ; CODE32 segment declared.
EXTRN MQCSNGL XTND: NEAR
; Assume DS and SS point to DATA (combined data/stack
; segment) where SNGL X and XTND X are declared.
: The following lines convert the single format
; number at SNGL X to an extended format number
: stored at XTND X.
                         ; EAX := offset(SNGL X)
LEA EAX, SNGL X
                         ; SNGL X offset
PUSH EAX
                         ; onto 80386 stack
                         : EAX := offset(XTND X)
LEA EAX, XTND X
                         : XTND X offset onto stack
PUSH EAX
CALL MQCSNGL XTND
```

; XTND\_X now contains the same value as SNGL\_X.

## mqcXTND DUBL

Converts number in extended format to same value in double format

#### **Parameters**

XTND\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to the input binary number in 80386 memory. If the input is not in 80-bit extended format, mqcXTND\_DUBL returns undefined output at DUBL\_PTR.

DUBL\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to the binary output buffer.

#### Discussion

mqcXTND\_DUBL converts a binary floating-point number in 80-bit extended format into the nearest number in 64-bit double format. mqcXTND\_DUBL rounds the input to the nearest value representable in double format, rounding to even if two double format values are equally near.

For the input 80387 special values ±∞ and ±0, mqcXTND\_DUBL returns equivalent representations in double format. For input NaNs, it truncates the least significant digits of the exponent (11..1) and significand to fit into double format, provided that all truncated significant digits are zeros.

## Exceptions

Possible exceptions are Invalid, Overflow, Underflow, and Precision:

I An input NaN whose significand bits to be truncated are not all zero causes the I exception. If I is masked, mqcXTND\_DUBL returns its truncated result at DUBL\_PTR if the truncated significand would be nonzero; when the truncated significand would be zero, it makes the result's significand nonzero by substituting the highest nonzero byte of the original significand into bits 5 through 12 of the significand at DUBL\_PTR. If I is unmasked, control passes to the exception handler with the input in ST (the 80387 stack top) and the exception opcode set.

## mqcXTND\_DUBL (continued)

- O An input exponent too large to fit in the double format exponent field causes the O exception. If O is masked, mqcXTND\_DUBL returns ± at DUBL\_PTR. If O is unmasked, control passes to the exception handler with the input value—its exponent unchanged but its significand rounded to double format—in ST and the exception opcode set.
- U A result too tiny to fit accurately in double format causes the U exception if U is masked. In this case, mqcXTND\_DUBL returns a gradual underflow denormal—rounded to fit in double format—at DUBL\_PTR. A result too tiny to be represented as a normalized double format number causes the U exception if U is unmasked. In this case, control passes to the exception handler with the input value—its exponent unchanged but its significand rounded to double format—in ST and the exception opcode set.
- P A result whose significand must be rounded to fit in double format causes the P exception. If P is masked, mqcXTND\_DUBL returns the rounded result at DUBL\_PTR. If P is unmasked, control passes to the exception handler with the input in ST and the exception opcode set.

## **Exception Opcode**

D5H

## mqcXTND\_DUBL (continued) Example

This example assumes that mqcXTND\_DUBL has not been declared.

```
EXTRN MOCXTND DUBL: FAR
             : declared outside all SEGMENT..ENDS
             : in module
            : Module's stack bytes allocated and
             : data segments declared here.
XTND ARY DT 8 DUP (?) ; 80-bit inputs
DUBL ARY DQ 8 DUP (?) ; 64-bit outputs
    : : SOMECODE USE32 segment declared here.
: Assume SS points to stack and DS points to
; segment where XTND ARY and DUBL ARY declared.
; The following lines convert extended format values
; at XTND ARY to double format values at DUBL ARY.
LEA EBX, DUBL ARY
                         : EBX := offset(DUBL ARY)
LEA EDX, XTND_ARY
                         ; EDX := offset(XTND ARY)
                         ; ECX := 8 (LOOP count req)
MOV ECX. 8
                         ; label
L00P8:
                        ; Save loop counter.
PUSH ECX
                        ; Save offset(DUBL ARY(n)).
PUSH EBX
                         ; Save offset(XTND ARY(n)).
PUSH EDX
PUSH DS
                         ; segment(XTND ARY) onto 80386
                         ; stack (4-byte push)
                         ; XTND ARY offset onto stack
PUSH EDX
                         ; segment(DUBL ARY) onto stack
PUSH DS
PUSH EBX
                         ; DUBL ARY offset onto stack
CALL MQCXTND DUBL
            : ESP now points to saved XTND ARY offset.
POP EDX
                         : Restore XTND ARY(n) offset.
ADD EDX. 10
                         : EDX := offset(XTND ARY(n+1))
POP EBX
                         : Restore DUBL ARY(n) offset.
ADD EBX. 8
                         ; EBX := offset(DUBL ARY(n+1))
                         ; Restore counter.
POP ECX
LOOP LOOP8
                          ; Decrement count in ECX.
                          ; Jump to LOOP8 if ECX not 0.
; DUBL ARY(0) through DUBL ARY(7) contain rounded
```

; copies of XTND ARY(0) through XTND ARY(7).

## mgcXTND SNGL

Converts number in extended format to same value in single format

## **Parameters**

XTND\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to the input binary number in 80386 memory. If the input is not in 80-bit extended format, mqcXTND\_SNGL returns undefined output at SNGL\_PTR.

SNGL\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to the binary output buffer.

#### Discussion

mqcXTND\_SNGL converts a binary floating-point number in 80-bit extended format into the nearest number in 32-bit single format. mqcXTND\_SNGL rounds the input to the nearest value representable in single format, rounding to even if two single format values are equally near.

For the input 80387 special values ±∞ and ±0, mqcXTND\_SNGL returns equivalent representations in single format. For input NaNs, it truncates the least significant digits of the exponent (11..1) and significand to fit into single format, provided that all truncated significant digits are zeros.

## **Exceptions**

Possible exceptions are Invalid, Overflow, Underflow, and Precision:

I An input NaN whose significand bits to be truncated are not all zero causes the I exception. If I is masked, mqcXTND\_SNGL returns its truncated result at SNGL\_PTR if the truncated significand would be nonzero; when the truncated significand would be zero, it makes the result's significand nonzero by substituting the highest nonzero byte of the original significand into bits 0 through 7 of the significand at SNGL\_PTR. If I is unmasked, control passes to the exception handler with the input in ST (the 80387 stack top) and the exception opcode set.

## mqcXTND SNGL (continued)

- O An input exponent too large to fit in the single format exponent field causes the O exception. If O is masked, mqcXTND\_SNGL returns ±00 at SNGL\_PTR. If O is unmasked, control passes to the exception handler with the input value—its exponent unchanged but its significand rounded to single format—in ST and the exception opcode set.
- U A result too tiny to fit accurately in single format causes the U exception if U is masked. In this case, mqcXTND\_SNGL returns a gradual underflow denormal—rounded to fit in single format—at SNGL\_PTR. A result too tiny to be represented as a normalized single format number causes the U exception if U is unmasked. In this case, control passes to the exception handler with the input value—its exponent unchanged but its significand rounded to single format—in ST and the exception opcode set.
- P A result whose significand must be rounded to fit in single format causes the P exception. If P is masked, mqcXTND\_SNGL returns the rounded result at SNGL\_PTR. If P is unmasked, control passes to the exception handler with the input in ST and the exception opcode set.

## **Exception Opcode**

D3H

; Store 80387 exception byte

: in 80386 memory.

## **Example**

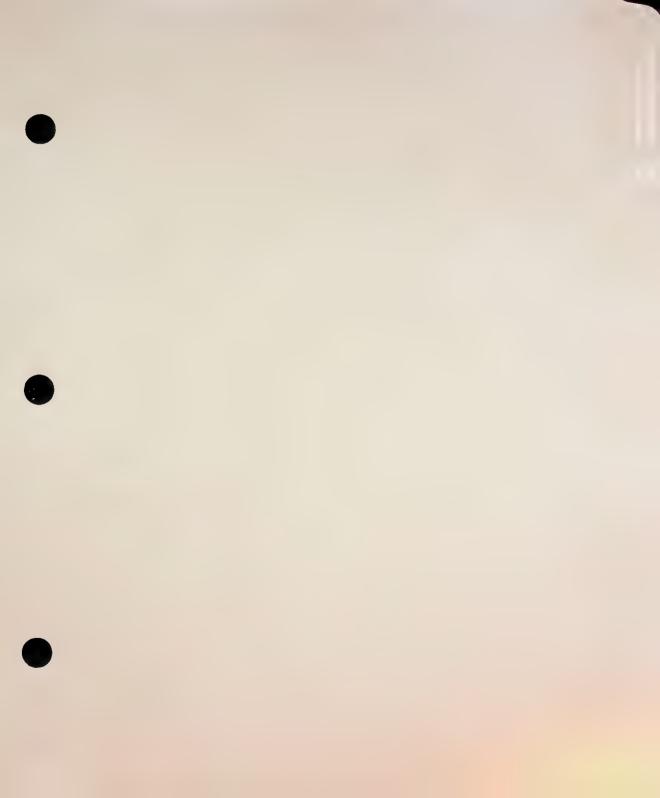
This example assumes that mqcXTND\_SNGL has not been declared. However, it assumes that exceptions are masked in the 80387 Control Word (see Section 2.1.6), so mqcXTND\_SNGL returns default results to SNGL\_VAR if exceptions occur; it also returns the 80387 exception byte to the AL register (see Sections 2.1.3 and 2.1.7).

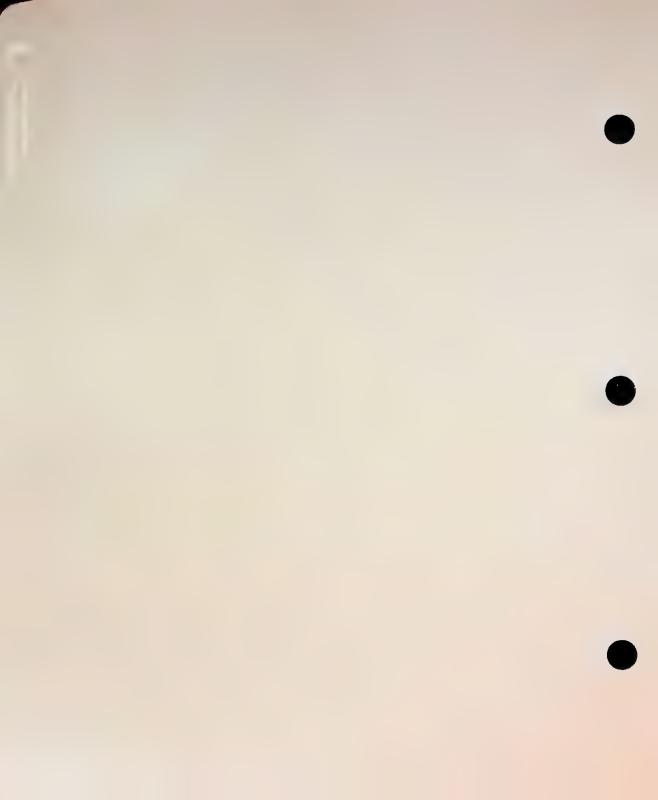
```
: DATA segment declared here.
XTND VAR DT ?
                          : 80-bit input
SNGL VAR DD ?
                          : 32-bit output
                          : for 80387 exception byte
XCPTNS DB 0
                          ; on return from MQCXTND SNGL
                          ; CODE32 segment declared.
EXTRN MOCXTND SNGL: NEAR
: Assume DS and SS point to DATA (combined data/stack
; segment) where XTND VAR and SNGL VAR are declared.
; The following lines convert the extended format
; number at XTND VAR to a single format number
; stored at SNGL VAR.
LEA EDX, XTND VAR
                          : EDX := offset(XTND VAR)
                          ; XTND VAR offset onto
PUSH EDX
                          : 80386 stack
LEA EDX, SNGL VAR
                          : EDX := offset(SNGL VAR)
PUSH EDX
                          : SNGL VAR offset onto stack
CALL MOCXTND SNGL
```

; SNGL\_VAR contains a rounded copy of the value at ; XTND\_VAR or a default result if exceptions occurred ; during MQCXTND\_SNGL's conversion. If the value in ; SNGL\_VAR is suspect, examine XCPTNS to see whether ; an 80387 exception bit was set (or which bit was ; set) during MQCXTND\_SNGL.

MOV XCPTNS. AL







## **Chapter 3 Common Elementary Function Library**

3.1	Libra	ry Overview	3-1
	3.1.1	Declaring CL387 Routines in ASM386 Programs	3-2
	3.1.2	CL387 Stack Requirements	.3-2
		3.1.2.1 80387 Stack Requirements	3-3
		3.1.2.2 80386 Stack Requirements	3-3
	3.1.3	CL387 Register Usage	3-3
	3.1.4	80387 Control Word Settings	3-4
	3.1.5	CL387 Numeric Exceptions	3-4
		3.1.5.1 CL387 Masked Exception Handling	3-6
		3.1.5.2 CL387 Unmasked Exception Handling	3-6
	3.1.6	CL387 Real and Complex Functions	3-8
		3.1.6.1 Special Values	3-8
		3.1.6.2 Machine_Pi	3-9
3.2	CL38	7 Real Functions	3-9
J	3.2.1	Summary of CL387 Real Functions3	-10
	3.2.2	How to Read the Real Function Reference Pages3	-12
	3.2.3	CL387 Real Function Reference3	-12
maei	ACS.	3	-13
maei	ASN.	3	-15
maei	ATN	3	-17
maei	COS.	3	-19
maei	rCSH.	3	-2
maei	DIM.	3	-23
maei	EXP.	3	-25
maei	IA2	3	-27
maei	IA4	3	-29
maei	rIA8	3	-3]
maei	IAX.	3	-33
moei	rIC2	3	-35
maei	rIC4	3	-37
maeı	rIC8	3	-39
maei	rICX.		-41
maei	IE2		-43
		3	
		3	

mgerMAX3-55
mgerMIN 3-57
mgerMOD3-59
mqerRMD3-61
mgerSGN
mqerSIN3-66
mqerSNH 3-68
mqerTAN3-70
mqerTNH3-72
mqerY2X 3-74
mqerYI23-78
mqerYI43-81
mqerY183-84
mqerYIS3-87
3.3 CL387 Complex Functions
3.3.1 Summary of CL387 Complex Functions 3-90
3.3.2 Complex Functions and Complex Numbers3-92
3.3.3 How to Read the Complex Function Reference Pages 3-94
3.3.4 CL387 Complex Function Reference
mqerCABS3-96
mqerCACH3-99
mqerCACS3-103
mqerCASH3-107
mqerCASN3-111
mqerCATH3-115
mqerCATN3-119
mqerCC2C 3-123
mqerCC2R 3-126
mqerCCI23-129
mqerCCI43-132
mqerCCI83-135
mqerCCIS3-138
mqerCCOS3-141
mqerCCSH3-144
mqerCDIV 3-147
mqerCEXP3-151
mqerCLGE 3-154
mqerCMUL 3-158
mqerCPOL3-161
mqerCPRJ3-165
mqerCR2C3-167
mgerCREC 3-170

mqerCSIN	3-173
mgerCSNH	3-176
mgerCSQR	3-179
mqerCTAN	3-181
mgerCTNH	3-184
Indered 11411	
Figures	
3-1 80387 Exception Byte for CL387	3-5
3-2 Rectangular Graph of a Complex Number	3-93
3-3 Polar Graph of a Complex Number	3-94
3-4 Absolute Value of a Complex Number	
3-5 Domain of Arccosh(z) with Branch Cuts	
3-6 Range of Arccosh(z)	3-100
3-7 Domain of Arccos(z) with Branch Cuts	3-104
3-8 Range of Arccos(z)	
3-9 Domain of Arcsinh(z) with Branch Cuts	
3-10 Range of Arcsinh(z)	3-108
3-11 Domain of Arcsin(z) with Branch Cuts	
3-12 Range of Arcsin(z)	3-112
3-13 Domain of Arctanh(z) with Branch Cuts	3-116
3-14 Range of Arctanh(z)	
3-15 Domain of Arctan(z) with Branch Cuts	
3-16 Range of Arctan(z)	
3-17 Domain of Ln(z) with Branch Cut	
3-18 Range of Ln(z)	
3-19 Rectangular and Polar Forms of a Complex Number	
3-20 Rectangular and Polar Forms of a Complex Number	
<b>4 0</b>	
Tables	
3-1 80387 State at CL387 Trap	3-7
3-2 Common Elementary Real Functions	
3-3 Common Elementary Complex Functions	



This chapter has three major sections:

- 1. An overview of the 80387 Common Elementary Function Library (CL387)
- 2. A reference for the CL387 real functions
- 3. A reference for the CL387 complex functions

## 3.1 Library Overview

The 80386-based 80387 Common Elementary Function Library consists of routines that return floating-point values. These functions can be linked with the OMF386 output of any Intel translator to execute on an 80387 or on an 80386 with a true software emulator.

CL387 library code is in ASM386 assembly language. CL387 extends. the set of functions provided by the 80387 both by adding functions and by allowing a wider range of input values for ASM386 floating-point instructions that it duplicates.

The following subsections explain:

- How to declare CL387 routines in ASM386 programs
- How much space the CL387 libraries need on the 80387 and 80386 stacks
- How CL387 uses the 80386 registers
- How the 80387 Control Word affects the CL387 routines and vice versa
- How CL387 detects, responds to, and reports 80387 exceptions
- How CL387 real functions differ from CL387 complex functions

## 3.1.1 Declaring CL387 Routines in ASM386 Programs

The Common Elementary Function Library can be linked to code that is within the same code segment or to code that is in another code segment. However, you must declare each CL387 routine that your program calls within each calling module. For example, within a module that calls mgerTAN, make one of the following declarations:

 If you want your code to be within the same segment as CL387, use the CL387N.LIB (near library). Declare mqerTAN as follows before calling this function:

CODE32 SEGMENT ER

; Segment name must be CODE32.

; CL387 segments are USE32.

EXTRN MQERTAN: NEAR

; MQERTAN is now callable.

: : CODE32 ENDS

• If you want your code to be in a different segment from CL387, use the CL387F.LIB (far library). Declare mqerTAN as follows before calling this function:

EXTRN MQERTAN: FAR

: Declaration must be

: outside all SEGMENT..ENDS

; pairs of the module.

; MQERTAN is now callable.

Declare only those CL387 routines that each module calls and link to the appropriate CL387N.LIB or CL387F.LIB file before executing the program. CL387N.LIB has a single combined data/stack segment named DATA. All CL387 library modules use 32-bit addressing. See the reference pages in Sections 3.2.3 and 3.3.4 for ASM386 examples of how to declare each CL387 routine. See Appendix B for more information about declaring CL387 routines in high-level language modules.

## 3.1.2 CL387 Stack Requirements

Most CL387 functions find their arguments and return results on the 80387 stack. Some functions also use the 80386 stack and registers. Sections 3.2.3 and 3.3.4 contain function-specific information about CL387 arguments and results.

#### 3.1.2.1 80387 Stack Requirements

CL387 requires at most four contiguous 80387 stack positions (registers) as working space and to hold arguments to its functions. For example, a complex function such as mqerCMUL requires the real and imaginary components of its arguments to be in the 80387 stack positions denoted by ST (stack top), ST(1), ST(2), and ST(3) (first-pushed argument). Therefore, ST(4), ST(5), ST(6), and ST(7) must be empty in order to push mqerCMUL's arguments without causing a stack overflow exception.

CL387 functions pop arguments and intermediate results from the 80387 stack before returning control to the caller. CL387 functions that return results to the 80387 stack leave only their results. All CL387 functions are completely reentrant: they use no fixed 80386 memory locations to store internal variables. However, any procedure that interrupts CL387 should save the entire 80387 stack because the stack elements a CL387 function is currently using cannot be identified.

## 3.1.2.2 80386 Stack Requirements

CL387N.LIB requires 224 bytes on the 80386 stack for its internal storage; CL387F.LIB requires 256 bytes. CL387 itself allocates the required stack bytes within its modules. However, a program such as a numeric exception handler could interrupt CL387 and then call CL387 again. If this occurs, the exception handler should allocate an additional 224 (near library) or 256 (far library) bytes on the 80386 stack at each recursion.

## 3.1.3 CL387 Register Usage

CL387 conforms to the register-saving conventions of Intel 80386 high-level languages. According to these conventions:

- The CL387 functions must leave the 80386 DS, ES, SS, and EBP registers unchanged. In fact, all CL387 functions except mqerYIS and mqerCIS also leave ESP unchanged. For CL387N.LIB, ES is assumed to access the same segment (DATA) as DS.
- The CL387 functions may destroy the contents of the EAX, EBX, ECX, EDX, EDI, ESI, FS, and GS registers.

## 3.1.4 80387 Control Word Settings

All CL387 functions save the 80387 Control Word upon entry and restore it upon exit.

All CL387 functions except mqerDIM compute results using the 80387 extended precision mode (64-bit significand) and changing the rounding mode as required by particular functions. For most functions, CL387 uses the 80387 default rounding mode: round to nearest with even preferred. mqerDIM uses the precision and rounding modes that are in effect when this function is called.

Programs that trap on the 80387 Precision (inexact) exception should set the 80387 precision mask before calling most CL387 functions; they should clear both the mask and its corresponding status bit to reinstate the trap after the call. A set precision bit can be a false positive for all but the following CL387 functions:

- mqerIAX, mqerIA2, mqerIA4, mqerIA8, mqerIEX, mqerIE2, mqerIE4, mqerIE8, mqerICX, mqerIC2, mqerIC4, mqerIC8
- mqerDIM
- mqerMOD, mqerRMD, mqerMAX, mqerMIN, mqerSGN

The CL387 functions maderIAX through maderIC8 convert real numbers to integer values. A set precision bit is a true positive for these functions and for maderDIM: P = 1 indicates that CL387 could not round or truncate a result to its nearest integer exactly. The CL387 functions maderMOD through maderSGN always return exact results.

CL387 constrains the use only of the precision mask. All other 80387 masks can be set according to the needs of your program.

## 3.1.5 CL387 Numeric Exceptions

CL387 functions signal the same numeric exceptions as the 80387 instructions. CL387 functions report exceptions by setting the appropriate bits in the 80387 exception byte. Figure 3-1 illustrates this low-order byte of the 80387's Status Word.

Bits:	7	6	5	4	3	2	1	0	
	Ε	S	P	U	0	Z	D	1	

Figure 3-1 80387 Exception Byte for CL387

The CL387 functions behave like atomic instructions in detecting, responding to, and reporting the I, D, Z, O, and U numeric exceptions:

- I CL387 signals the Invalid Operation exception for such programming errors as:
  - Input arguments that are outside the domain of a function
  - Input arguments in 80387 unsupported formats (pseudozeros, psuedo-NaNs, pseudoinfinities, and unnormals)
  - Not enough 80387 stack space for the function to execute (see Section 3.1.2.1). This exception also sets S (bit 6).
- D CL387 signals the Denormal exception for one or more denormal arguments.
- Z CL387 signals the Zerodivide exception when an operation on finite operands will produce, without overflow, an infinite result.
- O CL387 signals the numeric Overflow exception when the exponent of a rounded result is too large for the format of the destination.
- U CL387 signals the numeric Underflow exception when a result is too tiny to be represented accurately in the destination format if underflow is masked. It signals the Underflow exception when the absolute value of a result is less than the smallest positive normalized number if underflow is unmasked.

A CL387 function does not report its constituent instructions' I, D, Z, O, and U exceptions unless it should report the exceptions for particular arguments. For example, suppose masked numeric overflow occurs during an intermediate calculation and the resulting infinity becomes the divisor in a subsequent operation, producing a quotient of 0. Such a quotient could be a final result that is quite valid for a particular CL387 function applied to particular arguments. This function will suppress the O exception arising from its intermediate operation on these arguments.

CL387 functions report the other 80387 exceptions as follows:

- P For functions that convert floating-point numbers to integers and mqerDIM, CL387 signals the Precision exception when it cannot return an exact result (see Section 3.1.4).
- S CL387 signals a Stack overflow/underflow exception when there is not enough 80387 stack space for the function to operate (see Section 3.1.2.1). The I bit is also set when S is set.
- E CL387 sets the Exception status bit (7) if it sets any of the I, D, Z, O, U, and P bits (0..5) of the 80387 exception byte (see Figure 3-1).

## 3.1.5.1 CL387 Masked Exception Handling

CL387 responds to masked exceptions just as the 80387 does. It sets the appropriate bit of the 80387 exception byte and supplies a default result. See each function in Sections 3.2.3 and 3.3.4 for specific information about default results for masked exceptions.

# 3.1.5.2 CL387 Unmasked Exception Handling

CL387 provides exception handling information in the same data structure that the 80387 uses for handling unmasked exceptions: the 80387 State.

When an unmasked exception generates an Interrupt 16 (Processor Extension Error) from a CL387 function, the interrupt comes from a CL387 code site where cleanup of the 80386 stack is about to occur just before CL387's return to its caller. When an exception handler for the Processor Extension Error executes its IRETD, the return is to this cleanup section. The CL387 routine then returns control to its caller.

At the time of such a trap from CL387, the 80387 State consists of the 80387 Environment and 80387 stack information summarized in Table 3-1.

Table 3-1 80387 State at CL387 Trap

Field	Contains  caller's Control Word			
Control Word				
Status Word	(caller's Status Word OR CL387 exceptions)			
Tag Word	80387 stack, as shown (see STST(7) following)			
EIP Offset	FFFFFFFH for USE32 protected mode			
or Instruction Pointer	FFFFFFFH for USE32 real address mode			
Opcode	(((m + n) * 256) + CL387 function's exception opcode)			
where	m is one less than the number of function results			
	n is the number of arguments (1 or 2)			
ST	argument to CL387 function			
ST(1)	argument (functions with 2 or more arguments)			
ST(2) argument (functions with 3 or more arguments)				
ST(3) argument (functions with 4 arguments)				
ST(4)ST(7)	(inherited from CL387 function's caller)			

An exception handler for the Processor Extension Error can examine the 80387 Instruction Pointer or EIP Offset value to determine when CL387 generates an Interrupt 16. It can identify which CL387 function caused the interrupt by examining the Opcode field value (see Table 3-1).

CL387 functions handle all unmasked I, D, Z, O, and U exceptions as before-calculation errors: the original arguments are present on the 80387 stack at the time of the trap.

CL387 functions handle the unmasked P exception as an after-calculation error: the result of rounding its input real arguments is on the 80387 stack at the time of the trap. However, input arguments are not preserved.

## 3.1.6 CL387 Real and Complex Functions

The Common Elementary Function Library has two basic kinds of functions:

Real Functions accept real arguments and return real results

or integer results that are converted reals.

Complex Functions accept complex arguments and return complex

or real results.

Every CL387 real function begins with the prefix mqer. Every CL387 complex function begins with the prefix mqerC. The mqer prefix reduces the chance of conflict between a CL387 name and another name in your program. CL387 also has alternate PUBLIC names beginning with mqer for its routines. These names, listed in Appendix A, are used by some Intel translators.

To make function names more readable in this chapter, the mqer prefix is in lowercase letters, except in examples. However, uppercase and lowercase function names are interchangable within CL387 library modules.

## 3.1.6.1 Special Values

CL387 treats the special values  $\pm 0$ ,  $\pm \infty$ , and NaN (Not a Number) in the same manner as the 80387:

- +0 = -0 and square\_root(-0) = -0. The value zero is signed only for real and decimal integer formats. When a sum or difference of two operands with differing signs is exactly zero, all CL386 functions return +0 except for one function: mqerDIM returns -0 if the 80387 rounding mode is round down.
- -∞ < (any finite number) < +∞. The value infinity exists only for real formats. Infinity arithmetic is always exact, and the CL387 functions signal the Invalid exception only when an infinity argument is invalid for a particular function.
- NaNs are either signaling (SNaN) or quiet (QNaN). CL387 functions always report the Invalid exception for input SNaNs.
   They return QNaN indefinite results in the destination format for the Invalid exception when I is masked.

See Appendix D for the zero, infinity, and QNaN values in the 80387 formats used by CL387.

## 3.1.6.2 Machine Pi

True  $\pi$  is a transcendental number; it cannot be represented exactly by any rational number. Therefore, true  $\pi$  cannot be represented exactly in any floating-point format specified by the IEEE754 standard. It is possible to represent  $\pi$  only as accurately as a particular format will allow, rounded according to the current 80387 rounding mode.

However, no such value equals the 80387's machine\_pi that is used in all 80387 and CL387 trigonometric calculations. The 80387 machine\_pi has a 67-bit significand, while 80-bit extended format has a 64-bit significand. With machine\_pi, the 80387 trigonometric instructions produce extended format results that are accurate to 64 bits of precision. So do their CL387 extensions.

You might not get the period you expect for 80387 or CL387 trigonometric operations. For example, an extended format argument to FPTAN or mqerTAN will never be an exact odd multiple of machine\_pi/2. Therefore, FPTAN or mqerTAN never returns undefined results for the tangent function's asymptotes.

## 3.2 CL387 Real Functions

The real functions are a subset of the Common Elementary Function Library. This section contains:

- A summary of the CL387 real functions
- Information about how to read each function's reference pages
- A comprehensive reference for each CL387 real function in alphanumeric order

# 3.2.1 Summary of CL387 Real Functions

Table 3-2 summarizes the CL387 real functions.

Table 3-2 Common Elementary Real Functions

Compute Logarithms and Exponentials:			
Name	Function	Description	
mqerLGD	log <sub>in</sub> (x)	Common logarithm	
mgerLGE	ln(x)	Natural (base e) logarithm	
mqerEXP	e <sup>x</sup>	Exponential function	
mqerY2X	y <sup>x</sup> ory <sup>j</sup>	Raises y to real or integer power	
mqerYIS	y j	Raises y to power of 32-bit integer on 80386 stack	
mqerYl2	y <sup>j</sup>	Raises y to power of 16-bit integer in AX	
mqerYI4	y <sup>j</sup>	Raises y to power of 32-bit integer in EAX	
mqerYl8	y j	Raises y to power of 64-bit integer in EDX_EAX	

# Compute Trigonometrics and Hyperbolics:

Name	Function		Description
mqerSIN	sin(θ)		Trigonometric sine
mqerCOS	$cos(\theta)$		Trigonometric cosine
mqerTAN	tan(θ)		Trigonometric tangent
mqerASN	Arcsin(x)		Trigonometric inverse sine's principal value
mqerACS	Arccos(x)		Trigonometric inverse cosine's principal value
mqerATN	Arctan(x)		Trigonometric inverse tangent's principal value
mqerSNH	$sinh(\theta)$		Hyperbolic sine
mgerCSH	$cosh(\theta)$	•	Hyperbolic cosine
mqerTNH	tanh(θ)		Hyperbolic tangent

Table 3-2 Common Elementary Real Functions (continued)

Convert Rea	ls to I	ntegers:				
Nam	e .	Function	Description			
mqe	rIAX	roundaway(x)	Rounds x to nearest integer; rounds awa from 0 if two integers are equally near			
mqe	rlA2	roundaway(x)	Rounds x to 16-bit integer like maerIAX			
mqe	riA4	roundaway(x)	Rounds x to 32-bit integer like mqerlAX			
mqe	rIA8	roundaway(x)	Rounds x like to 64-bit integer like mqerIAX			
mqe	rIEX	roundeven(x)	Rounds x to nearest integer; rounds to even if two integers are equally near			
mqe	rlE2	roundeven(x)	Rounds x to nearest 16-bit integer like mgerIEX			
mqe	rIE4	roundeven(x)	Rounds x to nearest 32-bit integer like mgerIEX			
mqe	rIE8	roundeven(x)	Rounds x to nearest 64-bit integer like mgerIEX			
mqe	rICX	chop(x)	Truncates x to integer			
mqe	rlC2	chop(x)	Truncates x to 16-bit integer			
mqe	rlC4	chop(x)	Truncates x to 32-bit integer			
mqe	rlC8	chop(x)	Truncates x to 64-bit integer			
Return Other	er Valu	ies:				
Nam	e	Function	Description			
mqe	rMOD	x MOD y	Modulus retaining sign of x			
mqe	rRMD	x REM y	Remainder, rounding to nearest with ever preferred			
mqe	rMAX	max(x, y)	Greater of x or y			
mqe	rMIN	min(x, y)	Lesser of x or y			
mqe	rDIM	max(x - y, +0)	Positive difference			
mge	rSGN	x   with y's sign	Combines magnitude of x with sign of y			

## 3.2.2 How to Read the Real Function Reference Pages

The reference pages for each CL387 real function have five sections:

1. Function summarizes the result and input arguments for each function in two different ways. The first line uses 80387 stack position notation. Subsequent lines use mathematical notation.

For example,  $ST := ST(1)^{ST}$  indicates that a real input in ST(1) is raised to the power of a second real input in ST, and that the real result is left in ST.

The second line uses Result, y, and x to indicate the same thing mathematically, as Result :=  $y^x$ . This line and subsequent lines sometimes summarize restrictions on input values or further define the function.

- Discussion explains the function in more detail, including how it handles the special values ±∞ and ±0. This section uses the mathematical notation of the Function section. Ranges expressed as a..b are inclusive: a and b are part of the range.
- Exceptions first summarizes which exceptions the function can report. Then, it explains what causes each exception and how the function handles the masked and unmasked cases.
- 4. Exception Opcode has the hexadecimal value the function stores in the Opcode field of the 80387 State if the function generates a trap (see Table 3-1).
- 5. Example has a commented ASM386 example that shows how to declare the function for linkage with CL387N.LIB (see Section 3.1.1), how to set up the function's arguments (push order, register load, or both), how to call the function, and how to store the result.

## 3.2.3 CL387 Real Function Reference

The remaining pages in this section are a comprehensive reference for each Common Elementary Library real function. The functions are in alphanumeric order.

ST := Arccos(ST) Result := Arccos(x) if  $-1.0 \le x \le +1.0$ 

#### Discussion

mqerACS returns an angle whose trigonometric cosine equals the input x, where  $-1.0 \le x \le +1.0$ .

mqerACS returns the principal value of Arccos(x), expressed in radians. Results are in the range  $0..\pi$ . It returns  $\pi/2$  rounded to 64-bit precision for  $x = \pm 0$ .

## **Exceptions**

Possible exceptions are Invalid and Denormal:

- I Input x = ±∞, x = SNaN, x < -1.0, and x > +1.0 cause the I exception. If I is masked, mqerACS returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerACS returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## **Exception Opcode**

175H

# mqerACS (continued) Example

mqerACS returns Arccos(x) for x in the range -1.0..+1.0; its results fall in the range  $0..\pi$ .

```
; in DATA seament
HYPOTENUSE DQ 10.0 ; initialized ADJACENT SIDE DQ 5.0 ; to test value
                     to test values
ANGLE RADIANS DO ?
ANGLE DEGREES DO ?
RAD TO DEG DT 4004E52EE0D31E0FC2A9R; constant 180/pi
                        : in CODE32 segment
EXTRN MOERACS: NEAR
: The following lines compute the angle, given
: two sides of a right triangle.
                         : push, ST := ADJACENT SIDE
FLD ADJACENT SIDE
                        : ST := (ADJACENT SIDE /
FDIV HYPOTENUSE
                                      HYPOTENUSE)
                        ; ST := Arccos(ST)
CALL MOERACS
FST ANGLE RADIANS
                    ; Store result in memory.
FLD RAD TO DEG ; push, ST := 180/Pi
                        ST(1) := ST(1)*ST, pop ST
FMUL
FSTP ANGLE DEGREES ; Store result, pop stack.
: ANGLE DEGREES = 60 - It is a 30-60-90 triangle.
```

ST := Arcsin(ST) Result := Arcsin(x) if  $-1.0 \le x \le +1.0$ 

#### Discussion

mqerASN returns an angle whose trigonometric sine equals the input x, where  $-1.0 \le x \le +1.0$ .

mqerASN returns the principal value of Arcsin(x), expressed in radians. Results are in the range  $-\pi/2..+\pi/2$ . For  $x = \pm 0$ , mqerASN returns x unchanged.

## **Exceptions**

Possible exceptions are Invalid and Denormal:

- I Input  $x = \pm \infty$ , x = SNaN, x < -1.0, and x > +1.0 cause the I exception. If I is masked, mqerASN returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerASN returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## **Exception Opcode**

174H

# mqerASN (continued) Example

mqerASN returns Arcsin(x) for x in the range -1.0..+1.0; its results fall in the range  $-\pi/2..+\pi/2$ .

```
: in DATA segment
HYPOTENUSE DQ 10.0 ; initialized
OPPOSITE SIDE DO 5.0 : to test values
THETA DO ?
RAD TO DEG DT 4004E52EE0D31E0FC2A9R : constant 180/pi
                        : in CODE32 segment
EXTRN MOERASN: NEAR
; The following lines compute the angle, given
: two sides of a right triangle.
FLD OPPOSITE SIDE
                        : push, ST := OPPOSITE SIDE
FDIV HYPOTENŪSE
                        ; ST := (OPPOSITE SIDE /
                                    HYPOTENUSE)
CALL MOERASN
                       ; ST := Arcsin(ST)
FLD RAD TO DEG
                        ; push, ST := 180/pi
                       ; ST(1) := ST(1)*ST, pop ST
FMUL
FSTP THETA
                        : Store result, pop stack.
; THETA = 30 degrees - It is a 30-60-90 triangle.
```

ST := Arctan(ST) Result := Arctan(x) if  $-\infty \le x \le +\infty$ 

#### Discussion

mqerATN returns an angle whose trigonometric tangent equals the input x, where  $-\infty \le x \le +\infty$ .

mqerATN returns the principal value of Arctan(x), expressed in radians. Results are in the range  $-\pi/2..+\pi/2$ . For  $x = \pm 0$ , mqerATN returns x unchanged. It returns  $+\pi/2$  rounded to 64-bit precision for  $x = +\infty$  and  $-\pi/2$  rounded to 64-bit precision for  $x = -\infty$ .

## **Exceptions**

Possible exceptions are Invalid and Denormal:

- I Input SNaNs cause the I exception. If I is masked, mqerATN returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerATN returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## **Exception Opcode**

176H

# mqerATN (continued) Example

mqerATN returns Arctan(x) for x in the range  $-\infty..+\infty$ ; its results fall in the range  $-\pi/2..+\pi/2$ .

```
; in DATA segment
OPPOSITE SIDE DQ 5.0 ; initialized
ADJACENT SIDE DQ 5.0
                           : to test values
ANGLE RADIANS DQ ?
                           ; in CODE32 segment
EXTRN MQERATN: NEAR
; The following lines compute an angle, given
; two sides of a right triangle.
FLD OPPOSITE SIDE
                           ; push, ST := OPPOSITE SIDE
FLD OPPOSITE SIDE ; push, ST := OPPOSITE SI
FDIV ADJACENT_SIDE ; ST := (OPPOSITE_SIDE /
                                        ADJACENT SIDE)
CALL MOERATN
                         ; ST := Arctan(ST)
FSTP ANGLE RADIANS ; Store result, pop stack.
; ANGLE RADIANS = pi/4 - It is a 45-45-90 triangle.
```

ST: = cos(ST)Result:=  $cos(\theta)$  if  $-\infty < \theta < +\infty$ 

#### Discussion

mqerCOS returns the trigonometric cosine of an angle  $\theta$ , where  $-\infty < \theta < +\infty$  and  $\theta$  is expressed in radians. mqerCOS extends the range for  $\theta$  beyond the 80387 FCOS instruction's.

Results are in the range -1.0..+1.0. For  $\theta = \pm 0$ , mqerCOS returns +1.0.

# **Exception**

Possible exceptions are Invalid and Denormal:

- I Input  $\theta = \pm \infty$  and  $\theta = \text{SNaN}$  cause the I exception. If I is masked, mqerCOS returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input  $\theta$  still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCOS returns its result. If D is unmasked, control passes to the exception handler with  $\theta$  still in ST and the exception opcode set.

# **Exception Opcode**

172H

# mqerCOS (continued) Example

mqerCOS returns  $\cos(\theta)$  for  $\theta$  an angle in radians between but not including  $-\infty$  and  $+\infty$ ; its results fall in the range -1.0..+1.0.

```
; in DATA segment
ANGLE DEGREES DQ 30.0 ; initialized
RADIUS DO 2.0
                          : to test values
REC X DQ ?
DEG TO RAD DT 3FF98EFA351294E9C8AER
                          : constant pi/180
                          : in CODE32 segment
EXTRN MQERCOS: NEAR
: The following lines compute the X-coordinate
: of a polar-to-rectangular conversion.
FLD ANGLE DEGREES
                          : push. ST := ANGLE DEGREES
                          ; push, ST := DEG TO RAD
FLD DEG TO RAD
                          ; ST(1) := ST(1)*\overline{S}T, pop ST
FMUL
                          ; ST := cos(ST)
CALL MQERCOS
                         : ST := ST*RADIUS (Scale
FMUL RADIUS
                          : to correct radius.)
FSTP REC X
                          : Store X-coordinate.
                          : pop stack.
; REC X is now about square root(3).
```

ST :=  $\cosh(ST)$ Result :=  $\cosh(\theta)$  if  $-11355 \le \theta \le +11355$  or  $\theta = \pm \infty$ 

#### Discussion

mqerCSH returns the hyperbolic cosine of an angle  $\theta$ , where -11355  $\leq \theta \leq +11355$  or  $\theta = \pm \infty$ .

Results are in the range  $+1.0..+\infty$ . For  $\theta=\pm 0$ , mqerCSH returns +1.0; for  $\theta=\pm \infty$ , it returns  $+\infty$ .

# **Exceptions**

Possible exceptions are Invalid, Denormal, and Overflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCSH returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input  $\theta$  still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCSH returns its result. If D is unmasked, control passes to the exception handler with  $\theta$  still in ST and the exception opcode set.
- O Input  $\theta$  outside the range -11355..+11355 but not equal to  $\pm \infty$  cause the O exception. If O is masked, mqerCSH returns  $-\infty$  for  $\theta$  < -11355 and  $+\infty$  for  $\theta$  > +11355. If O is unmasked, control passes to the exception handler with  $\theta$  still in ST and the exception opcode set.

# **Exception Opcode**

16FH

# mqerCSH (continued) Example

mqerCSH returns  $cosh(\theta)$ ; its results fall in the range +1.0..+ $\infty$ .

; in DATA segment

THETA DO 1.5 : initialized to test value

COSH VALUE DQ ?

: ; in CODE32 segment

EXTRN MQERCSH: NEAR

; The following lines compute cosh(THETA).

FLD THETA ; ST := THETA CALL MQERCSH ; ST := cosh(ST)

FSTP COSH VALUE : Store result, pop stack.

; COSH VALUE now is about 2.3524096.

ST :=  $\max(ST(1) - ST, +0)$ Result :=  $\max(x - y, 0)$  if  $-\infty \le x \le +\infty$  and  $-\infty \le y \le +\infty$  as follows:

- Result := (x y) if  $x \ge y$
- Result := 0 if x < y

#### Discussion

mqerDIM returns (x - y) where x and y are numbers if their difference is not less than zero. It returns 0 if (x - y) < 0. mqerDIM uses the 80387 precision and rounding modes that are in effect when it is called. As a conditional subtraction, its results vary according to your settings for the precision and rounding bits of the 80387 Control Word; when both operands are  $\pm 0$ , it is possible to get a -0 result.

mqerDIM accepts  $x = \pm \infty$  or  $y = \pm \infty$  as long as  $x \neq y$ , and returns:

- +0 if  $x = -\infty$  and  $-\infty < y \le +\infty$
- $+\infty$  if  $x = +\infty$  and  $-\infty \le y < +\infty$
- +0 if  $-\infty \le x < +\infty$  and  $y = +\infty$
- $+\infty$  if  $-\infty < x \le +\infty$  and  $y = -\infty$

# **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Precision:

- I Input  $x = y = +\infty$ ,  $x = y = -\infty$ , and (x or y) = SNaN cause the I exception. If I is masked, mqerDIM returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST(1), y still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerDIM returns its result. If D is unmasked, control passes to the exception handler with x still in ST(1), y still in ST, and the exception opcode set.

## mgerDIM (continued)

- O The subtraction (x y) where x > y can cause the O exception. If O is masked, mqerDIM returns  $+\infty$ . If O is unmasked, mqerDIM stores its result in ST, adjusting the out-of-range exponent to fit in the extended format exponent field. mqerDIM subtracts the constant 24576 (decimal) to make this adjustment. Then, control passes to the exception handler with the exception opcode set.
- P An inexact (x y) causes the P exception. If P is masked, mqerDIM returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

## **Exception Opcode**

265H

## Example

mqerDIM returns the greater of (x - y) and +0 where x and y are in the range  $-\infty..+\infty$  but are not equivalent infinities.

```
: in DATA segment
COSTS DQ 650.00
                        ; initialized
RECEIPTS DQ 800.00 ; to test values
PROFIT DO ?
LOSS DO ?
                         ; in CODE32 segment
EXTRN MQERDIM: NEAR
: The following lines store the positive difference
: in PROFIT or LOSS and set the other value to zero.
FLD RECEIPTS
                         ; push, ST := RECEIPTS
FLD COSTS
                         ; push, ST := COSTS
CALL MQERDIM
                         ; ST := max(ST(1) - ST, +0)
FSTP PROFIT
                         ; Store result, pop stack.
            ; Now perform function the other way.
FLD COSTS
                         ; push, ST := COSTS
FLD RECEIPTS
                         ; push, ST := RECEIPTS
CALL MOERDIM
                         ; ST := max(ST(1) - ST, +0)
FSTP LOSS
                         ; Store result, pop stack.
```

ST := 
$$e^{ST}$$
  
Results :=  $e^{x}$  if  $-4112 \le x \le +11356.52$  or  $x = \pm \infty$ 

#### Discussion

mqerEXP returns the value of e (the transcendental constant 2.71828182845904523536..) raised to the power of the input number x, where  $-4112 \le x \le +11356.52$  or  $x = \pm \infty$ .

Results of the exponential function are in the range +0..+... mqerEXP returns:

- 1.0 for  $x = \pm 0$
- +0 for  $x = -\infty$
- $+\infty$  for  $x = +\infty$

## **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerEXP returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerEXP returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.
- O Input +11356.52 < x < +∞ cause the O exception. If O is masked, mqerEXP returns +∞. If O is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.
- U Input  $-\infty < x < -4112$  cause the U exception. If U is masked, mqerEXP returns a gradual underflow denormal, if possible, and  $\pm 0$  otherwise. If U is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

# mqerEXP (continued) Exception Opcode

16BH

## Example

mqerEXP returns  $e^x$  where x is in the range -4112..+11356.52 or is  $\pm \infty$ ; its results fall in the range +0..+ $\infty$ .

```
; in DATA segment
                    ; initialized to test value
X VALUE DQ 0.4
                     ; constant
MĪNUS 2 DQ -2
Y VALUE DQ ?
NORM CONSTANT DT 3FFDCC42299EA1B28697R
                         ; 1/square root(2*pi)
                         ; in CODE32 segment
EXTRN MQEREXP: NEAR
; The following lines compute the normal distribution.
FLD X VALUE
                         ; push, ST := X VALUE
FMUL ST,ST
                        ; ST := X VALUE**2
FIDIV MINUS 2
                        ; ST := S\overline{T} / -2
CALL MOEREXP
                        ; ST. := e**ST
                         ; push, ST := NORM CONSTANT
FLD NORM_CONSTANT
                        ; (ST = 1 / square root(2*pi))
FMUL
                         ; ST(1) := ST(1)*S\overline{T}, pop ST
FSTP Y VALUE
                         : Store result, pop stack.
; Y VALUE is now about 0.368270140.
```

Returns nearest word integer for number rounded away from zero

# Function

AX := roundaway(ST)
Result := roundaway(x) as follows:

- Result := round to nearest(x) if fraction part(x)  $\neq$  0.5
- Result := (x + 0.5) if fraction\_part(x) = 0.5 and  $x \ge 0$
- Result := (x 0.5) if fraction\_part(x) = 0.5 and x < 0

#### Discussion

mqerIA2 returns the nearest 16-bit integer for the input number x, where -32,768.5 < x < +32,767.5. Results are in the range -32768..+32767, expressed in two's complement.

When x falls midway between two integers, mqerIA2 rounds x away from zero, returning the result with the larger absolute value. For example, mqerIA2 returns:

- 0003H = 3 for x = 3.1
- 000BH = 11 for x = 10.5
- FFFDH for x = -2.5; FFFDH = -3 in word integer format

## **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

- I Input numbers that cannot fit the 16-bit destination after rounding (x ≤ -32768.5 or x ≥ +32767.5) cause the I exception, as do infinities and SNaNs. If I is masked, mqerIA2 returns the 16-bit indefinite integer (8000H = -32768). If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerIA2 returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## mgerIA2 (continued)

P Input x values that are not integers cause the P exception. If P is masked, mqerIA2 returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

## **Exception Opcode**

17EH

## Example

mqerIA2 returns the nearest word integer for an input number between but not including -32768.5 and +32767.5; its results fall in the range -32768..+32767. If the input is equally close to two integer values, mqerIA2 returns the result with the larger absolute value.

mgerIA4

Returns nearest short integer for number rounded away from zero

## **Function**

EAX := roundaway(ST)
Result := roundaway(x) as follows:

- Result := round\_to\_nearest(x) if fraction\_part(x) ≠ 0.5
- Result := (x + 0.5) if fraction part(x) = 0.5 and  $x \ge 0$
- Result := (x 0.5) if fraction\_part(x) = 0.5 and x < 0

#### Discussion

mqerIA4 returns the nearest 32-bit integer for the input number x, where -2,147,483,648.5 < x < +2,147,483,647.5 ( $-2^{31} - 0.5 < x < +2^{31} - 0.5$ ). Results are in the range  $-2^{31}..+(2^{31} - 1)$ , expressed in two's complement.

When x falls midway between two integers, mqerIA4 rounds x away from zero, returning the result with the larger absolute value. For example, mqerIA4 returns:

- 00000003H = 3 for x = 3.1
- 0000000BH = 11 for x = 10.5
- FFFFFFDH for x = -2.5; FFFFFFDH = -3 in short integer format

# **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

I Input numbers that cannot fit the 32-bit destination after rounding ( $x \le -2147483648.5$  or  $x \ge +2147483647.5$ ) cause the I exception, as do infinities and SNaNs. If I is masked, magrIA4 returns the 32-bit indefinite integer (80000000H =  $-2^{31}$ ). If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.

## mgerIA4 (continued)

- D Input denormals cause the D exception. If D is masked, mqerIA4 returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.
- P Input x values that are not integers cause the P exception. If P is masked, mqerIA4 returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

## **Exception Opcode**

168H

## Example

mqerIA4 returns the nearest short integer for an input number between but not including  $(-2^{31} - 0.5)$  and  $(2^{31} - 0.5)$ ; its results fall in the range  $-2^{31}$ ..+ $(2^{31} - 1)$ . If the input is equally close to two integer values, mqerIA4 returns the result with the larger absolute value.

```
; in DATA segment
REAL COUNT DQ 100000.5; initialized to test value
COUNT DD?

; in CODE32 segment
EXTRN MQERIA4: NEAR

; The following lines round REAL COUNT to the nearest
; integer, rounding away from zero to the larger
; absolute value when two integers are equally near.

FLD REAL COUNT
CALL MQERIA4
MOV COUNT, EAX
; Store result.

; COUNT is now 100001 (000186A1H).
```

Returns nearest long integer for number rounded away from zero

## **Function**

EDX\_EAX := roundaway(ST)
Result := roundaway(x) as follows:

- Result := round to nearest(x) if fraction part(x)  $\neq$  0.5
- Result := (x + 0.5) if fraction\_part(x) = 0.5 and  $x \ge 0$
- Result := (x 0.5) if fraction\_part(x) = 0.5 and x < 0

#### Discussion

mqerIA8 returns the nearest 64-bit integer for the input number x, where  $(-2^{63} - 0.5) < x < +(2^{63} - 0.5)$ . Results are in the range  $-2^{63}..+(2^{63} - 1)$ , expressed in two's complement.

When x falls midway between two integers, mqerIA8 rounds x away from zero, returning the result with the larger absolute value. For example, mqerIA8 returns:

- $00000000 \ 00000003H = 3 \ \text{for } x = 3.1$
- $00000000 \ 00000000BH = 11 \ for \ x = 10.5$
- FFFFFFF FFFFFFDH for x = -2.5; FFFFFFFF FFFFFFDH =
   -3 in long integer format

## **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

- Input numbers that cannot fit the 64-bit destination after rounding ( $x \le -2^{63} 0.5$  or  $x \ge +2^{63} 0.5$ ) cause the I exception, as do infinities and SNaNs. If I is masked, mqerIA2 returns the 64-bit indefinite integer ( $800000000000000000 = -2^{63}$ ). If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerIA8 returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## mgerIA8 (continued)

P Input x values that are not integers cause the P exception. If P is masked, mqerIA8 returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

## **Exception Opcode**

185H

## Example

mqerIA8 returns the nearest long integer for an input number between but not including  $(-2^{63} - 0.5)$  and  $(2^{63} - 0.5)$ ; its results fall in the range  $-2^{63}$ ..+ $(2^{63} - 1)$ . If the input is equally close to two integer values, mqerIA8 returns the result with the larger absolute value.

```
; in DATA segment
REAL VAR DQ 8.5
                      : initialized to test value
INT VAR DO ?
                         ; in CODE32 segment
EXTRN MOERIA8: NEAR
; The following lines round REAL VAR to the nearest
; integer, rounding away from zero to the larger
: absolute value if two integers are equally near.
FLD REAL VAR
                         : push. ST := REAL VAR
CALL MOERIA8
                         : EDX EAX := roundaway(ST)
                         : EAX := lower 4 bytes(ST)
                         ; EDX := next 4 bytes(ST)
MOV INT_VAR, EAX
                         : Store least significant
                         : digits of INT VAR.
MOV INT VAR+4, EDX
                         : Store most significant
                         ; digits of INT VAR.
; INT VAR is now 9 (00000000 00000009H).
```

Returns nearest integer for number rounded away from zero

# **Function**

ST := roundaway(ST)
Result := roundaway(x) as follows:

- Result := round\_to\_nearest(x) if fraction\_part(x) \neq 0.5
- Result := (x + 0.5) if fraction\_part(x) = 0.5 and  $x \ge 0$
- Result := (x 0.5) if fraction\_part(x) = 0.5 and x < 0

#### Discussion

mqerIAX returns the nearest integer for the input number x, where  $-\infty \le x \le +\infty$ . Results fall in the same range and remain in 80-bit extended format.

When x falls midway between two integers, mqerIAX rounds x away from zero, returning the result with the larger absolute value. For example, mqerIAX returns:

- 3 for x = 3.3
- 5 for x = 4.5
- -7 for x = -6.5

For  $x = \pm \infty$ , mqerIAX returns x unchanged.

# **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

- I Input SNaNs cause the I exception. If I is masked, mqerIAX returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerIAX returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## mgerIAX (continued)

P Input x values that are not integers cause the P exception. If P is masked, mqerIAX returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

## **Exception Opcode**

167H

## Example

mqerIAX returns the nearest integer for an input number in the range  $-\infty..+\infty$ ; its results fall in the same range. If the input is equally close to two integer values, mqerIAX returns the result with the larger absolute value.

```
: in DATA segment
HOURS DQ 2.71
                        ; initialized to test value
SIXTY DD 60.0
                          : constant
MINUTES DT ?
                          ; in CODE32 segment
EXTRN MQERIAX: NEAR
; The following lines convert HOURS into the nearest
; integer number of MINUTES, rounding away from zero
; to the larger absolute value for MINUTES
: if two integers are equally near.
FLD HOURS
                          ; push, ST := HOURS
FMUL SIXTY
                          : ST := ST*60
                          ; ST := roundaway(ST)
CALL MOERIAX
FSTP MINUTES
                          ; Store result, pop stack.
; MINUTES is now 163.
```

AX := chop(ST) Result := chop(x) as follows:

• Result := x - (fraction part(| x |) with sign of x)

#### Discussion

mqerIC2 returns the nearest 16-bit integer for the input number x by truncating any fraction part of x. Results are in the range -32768..+32767, expressed in two's complement.

In effect, mqerIC2 rounds a real x towards zero when -32,769 < x < +32,768. For example, mqerIC2 returns:

- 0004H = 4 for x = 4.99999999
- 000BH = 11 for x = 11.7
- FFFAH for x = -6.9; FFFAH = -6 in word integer format

## **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

- I Input numbers that cannot fit the 16-bit destination after truncation ( $x \le -32769$  or  $x \ge +32768$ ) cause the I exception, as do infinities and SNaNs. If I is masked, mqerIC2 returns the 16-bit indefinite integer (8000H = -32768). If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerIC2 returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.
- P Input x values that are not integers cause the P exception. If P is masked, mqerIC2 returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

# mqerIC2 (continued) Exception Opcode

17FH

## Example

mqerIC2 returns the nearest word integer for an input number that falls between but not including -32769 and +32768; its results fall in the range -32768..+32767.

```
: : ; in DATA segment

REAL_INPUT_DQ 65.7 ; initialized to test value
CONTROL_SETTING_DW 0 ; initialized

: : ; in CODE32 segment

EXTRN MQERIC2: NEAR

; The following lines convert the REAL_INPUT
; into the 16-bit integer value CONTROL_SETTING
; by truncating any fraction part of REAL_INPUT.

FLD REAL_INPUT ; push, ST := REAL_INPUT
CALL_MQERIC2 ; AX := chop(ST)
MOV_CONTROL_SETTING,AX ; Store result.

; CONTROL_SETTING is now 65 (41H).
```

Returns short integer for truncated number

## **Function**

EAX := chop(ST) Result := chop(x) as follows:

Result := x - (fraction part(| x |) with sign of x)

#### Discussion

mqerIC4 returns the nearest 32-bit integer for the input number x by truncating any fraction part of x. Results are in the range  $-2^{31}$ ..+ $(2^{31} - 1)$ , expressed in two's complement.

In effect, mqerIC4 rounds a real x towards zero when  $-2,147,483,649 < x < +2,147,483648 (<math>-2^{31} - 1 < x < +2^{31}$ ). For example, mqerIC4 returns:

- 00000004H = 4 for x = 4.9999999
- 0000000BH = 11 for x = 11.7
- FFFFFFAH for x = -6.9; FFFFFFAH = -6 in short integer format

## **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

- I Input numbers that cannot fit the 32-bit destination after truncation ( $x \le -2147483649$  or  $x \ge +2147483648$ ) cause the I exception, as do infinities and SNaNs. If I is masked, mqerIC4 returns the 32-bit indefinite integer ( $80000000H = -2^{31}$ ). If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerIC4 returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## mqerIC4 (continued)

P Input x values that are not integers cause the P exception. If P is masked, mqerIC4 returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

# **Exception Opcode**

179H

## Example

mqerIC4 returns the nearest short integer for an input number that falls between but not including  $-2^{31} - 1$  and  $+2^{31}$ ; its results fall in the range  $-2^{31}...+(2^{31} - 1)$ .

```
; : ; in DATA segment
VOTERS DQ 5306279.0 ; initialized to test values
SUPPORT SHARE DQ 0.39 ; (estimated % supporters)
VOTES DD ?

; in CODE32 segment
EXTRN MQERIC4: NEAR

; The following lines calculate the estimated number
; of votes for a particular issue, truncating any
; fraction part of VOTERS*SUPPORT_SHARE.

FLD VOTERS ; push, ST := VOTERS
FMUL SUPPORT_SHARE ; ST := ST*SUPPORT_SHARE
CALL MQERIC4 ; EAX := chop(ST)
MOV VOTES, EAX ; Store result.

; VOTES is now 2069448 (001F93C8H).
```

EDX\_EAX := chop(ST)
Result := chop(x) as follows:

• Result := x - (fraction part(| x !) with sign of x)

#### Discussion

mqerIC8 returns the nearest 64-bit integer for the input number x by truncating any fraction part of x. Results are in the range  $-2^{63}$ ..+ $(2^{63} - 1)$ , expressed in two's complement.

In effect, mqerIC8 rounds a real x towards zero when  $-2^{63} - 1 < x < +2^{63}$ . For example, mqerIC8 returns:

- $00000000\ 00000004H = 4 \text{ for } x = 4.9999999$
- $00000000 \ 0000000BH = 11 \ for \ x = 11.7$
- FFFFFFF FFFFFFAH for x = -6.9; FFFFFFFF FFFFFFAH =
   -6 in long integer format

## **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

- I Input numbers that cannot fit the 64-bit destination after truncation ( $x \le -(2^{63} + 1)$  or  $x \ge +2^{63}$ ) cause the I exception, as do infinities and SNaNs. If I is masked, mqerIC8 returns the 64-bit indefinite integer (8000000000000000H =  $-2^{63}$ ). If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerIC8 returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## mgerIC8 (continued)

P Input x values that are not integers cause the P exception. If P is masked, mqerIC8 returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

## **Exception Opcode**

184H

## Example

mqerIC8 returns the nearest long integer for an input number that falls between but not including  $-2^{63} - 1$  and  $+2^{63}$ ; its results fall in the range  $-2^{63}...+(2^{63}-1)$ .

```
; in DATA segment
VOTERS DQ 53062790.0 ; initialized to test values
SUPPORT SHARE DQ 0.39 ; (estimated % supporters)
VOTES DO ?
                         : in CODE32 segment
EXTRN MQERIC8: NEAR
: The following lines calculate the estimated number
; of votes for a particular issue, truncating any
; fraction part of VOTERS*SUPPORT SHARE.
                         : push, ST := VOTERS
FLD VOTERS
FMUL SUPPORT SHARE
                         : ST := ST*SUPPORT SHARE
CALL MOERIC8
                         ; EDX EAX := chop(\overline{S}T)
                         ; EAX := lower 4 bytes(ST)
                         ; EDX := next 4 bytes(ST)
MOV VOTES EAX
                         : Store least significant
                         ; digits of VOTES.
MOV VOTES+4, EDX
                         : Store most significant
                          : digits of VOTES.
; VOTES is now 20694488 (00000000 0136C5D8H).
```

### **Function**

ST := chop(ST) Result := chop(x) as follows:

• Result := x - (fraction part(| x |) with sign of x)

#### Discussion

mqerICX returns the nearest integer for the input number x by truncating any fraction part of x. Results fall in the range  $-\infty \le x \le +\infty$  and remain in 80-bit extended format.

In effect, mqerICX rounds a real x towards zero when  $-\infty \le x \le +\infty$ . For example, mqerICX returns:

- 1 for x = 1.7
- -3 for x = -3.9

For  $x = \pm \infty$ , mqerICX returns x unchanged.

### **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

- I Input SNaNs cause the I exception. If I is masked, mqerICX returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerICX returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.
- P Input x values that are not integers cause the P exception. If P is masked, mqerICX returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

# mqerICX (continued) Exception Opcode

166H

### Example

mqerICX returns the nearest integer for an input number in the range  $-\infty..+\infty$ ; its results fall in the same range.

```
; in DATA segment
PRICE DOLLARS DT 37.596; initialized to test value
ONE HUNDRED DD 100.00 ; constant
                          ; in CODE32 segment
EXTRN MQERICX: NEAR
; The following lines chop PRICE DOLLARS to a discrete
; number of pennies by truncating any fraction
; of a penny.
FLD PRICE DOLLARS
                        ; push, ST := PRICE DOLLARS
                      ; ST := ST*100
FMUL ONE HUNDRED
CALL MOERICX
                         ; ST := chop(ST)
FDIV ONE HUNDRED ; ST := ST/100
FSTP PRICE_DOLLARS ; Store result, pop stack.
; PRICE DOLLARS is now 37.59.
```

# mgerlE2

Returns word integer for number rounded to nearest with even preferred

# Function

AX := roundeven(ST)
Result := roundeven(x) as follows:

- Result := round to nearest(x) if fraction part(x)  $\neq$  0.5
- Result := (x + 0.5) if fraction\_part(x) = 0.5 and (x + 0.5) is even
- Result := (x 0.5) if fraction\_part(x) = 0.5 and (x + 0.5) is odd

#### Discussion

mqerIE2 returns the nearest 16-bit integer for input number x, where  $-32,768.5 \le x < +32,767.5$ . Results are in the range -32768..+32767, expressed in two's complement.

When x falls midway between two integers, mqerIE2 rounds x to the even integer value. For example, mqerIE2 returns:

- 0003H = 3 for x = 3.1
- 000AH = 10 for x = 10.5
- FFFEH for x = -2.5; FFFEH = -2 in word integer format

## **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

- I Numbers that cannot fit the 16-bit destination (x < -32768.5 or x ≥ +32767.5) cause the I exception, as do infinities and SNaNs. If I is masked, mqerIE2 returns the 16-bit indefinite integer (8000H = -32768). If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.</p>
- D Input denormals cause the D exception. If D is masked, mqerIE2 returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## mgerlE2 (continued)

P Input x values that are not integers cause the P exception. If P is masked, mqerIE2 returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

## **Exception Opcode**

180H

## Example

mqerIE2 returns the nearest word integer for an input number x in the range  $-32768.5 \le x < +32767.5$ ; its results fall in the range -32768.32767.

```
: : ; in DATA segment

REAL_VAR_DQ -8.5 ; initialized to test value

INT_VAR_DW ?

: : ; in CODE32 segment

EXTRN MQERIE2: NEAR

; The following lines round REAL_VAR to the nearest
; word integer, rounding to the even integer
; when REAL_VAR's fraction part equals 0.5.

FLD REAL_VAR

; push, ST := REAL_VAR

CALL_MQERIE2
; AX := roundeven(ST)

MOV_INT_VAR,AX
; Store result.
```

; INT VAR now equals -8 (FFF8H).

Returns short integer for number rounded to nearest with even preferred

## **Function**

EAX = roundeven(ST)
Result := roundeven(x) as follows:

- Result := round to nearest(x) if fraction part(x)  $\neq$  0.5
- Result := (x + 0.5) if fraction part(x) = 0.5 and (x + 0.5) is even
- Result := (x 0.5) if fraction part(x) = 0.5 and (x + 0.5) is odd

#### Discussion

mqerIE4 returns the nearest 32-bit integer for input number x, where  $-2,147,483,648.5 \le x < 2,147,483,647.5 (-2^{31} - 0.5 \le x < 2^{31} - 0.5)$ . Results are in the range  $-2^{31}..+(2^{31} - 1)$ , expressed in two's complement.

When x falls midway between two integers, mqerIE4 rounds x to the even integer value. For example, mqerIE4 returns:

- 00000003H = 3 for x = 3.1
- 0000000AH = 10 for x = 10.5
- FFFFFFEH for x = -2.5; FFFFFFEH = -2 in short integer format

## **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

- I Numbers that cannot fit the 32-bit destination (x < -2147483648.5 or x  $\ge$  +2147483647.5) cause the I exception, as do infinities and SNaNs. If I is masked, magerIE4 returns the 32-bit indefinite integer (80000000H =  $-2^{31}$ ). If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerIE4 returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## mgerIE4 (continued)

P Input x values that are not integers cause the P exception. If P is masked, mqerIE4 returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

## **Exception Opcode**

17BH

### Example

mqerIE4 returns the nearest short integer for an input number x in the range  $-(2^{31} + 0.5) \le x < +(2^{31} - 0.5)$ ; its results fall in the range  $-2^{31}..+(2^{31} - 1)$ .

```
: : : in DATA segment
REAL_COUNT DQ 100000.5 ; initialized to test value
COUNT DD ?

: : ; in CODE32 segment
EXTRN MQERIE4: NEAR
```

; The following lines round REAL\_COUNT to the nearest ; short integer, rounding to the even integer

; if REAL COUNT's fraction part equals 0.5.

FLD REAL\_COUNT

CALL MQERIE4

MOV COUNT, EAX

; push, ST := REAL\_COUNT

EAX := roundeven(ST)

Store result.

; COUNT is now 100000 (000186A0H).

mgerIE8

Returns long integer for number rounded to nearest with even preferred

# **Function**

EDX\_EAX := roundeven(ST)
Result := roundeven(x) as follows:

- Result := round to nearest(x) if fraction part(x)  $\neq$  0.5
- Result := (x + 0.5) if fraction part(x) = 0.5 and (x + 0.5) is even
- Result := (x 0.5) if fraction\_part(x) = 0.5 and (x + 0.5) is odd

#### Discussion

mqerIE8 returns the nearest 64-bit integer for input number x where  $-(2^{63} + 0.5) \le x < +(2^{63} - 0.5)$ . Results are in the range  $-2^{63}...+(2^{63} - 1)$ , expressed in two's complement.

When x falls midway between two integers, mqerIE8 rounds x to the even integer value. For example, mqerIE8 returns:

- $00000000 \ 00000003H = 3 \ \text{for } x = 3.1$
- $00000000 \ 0000000AH = 10 \ for \ x = 10.5$
- FFFFFFF FFFFFFF FFFFFFFF FFFFFFFF = -2 in long integer format

## Exceptions

Possible exceptions are Invalid, Denormal, and Precision:

- D Input denormals cause the D exception. If D is masked, mqerIE8 returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

### mgerIE8 (continued)

P Input x values that are not integers cause the P exception. If P is masked, mqerIE8 returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

## **Exception Opcode**

186H

# Example

mqerIE8 returns the nearest long integer for an input number x in the range  $-(2^{63} + 0.5) \le x < +(2^{63} - 0.5)$ ; its results fall in the range  $-2^{63}..+(2^{63} - 1)$ .

```
: in DATA segment
: :
REAL VAR DQ 8.5
                      : initialized to test value
INT VAR DQ ?
                         : in CODE32 segment
EXTRN MOERIE8: NEAR
: The following lines round REAL VAR to the nearest
; long integer, rounding to the even integer
: if REAL VAR's fraction part equals 0.5.
FLD REAL VAR
                         : push. ST := REAL VAR
CALL MOERIES
                        : EDX EAX := roundeven(ST)
                         : EAX := lower 4 bytes(ST)
                       ; EDX := next 4 bytes(ST)
MOV INT VAR, EAX
                         : Store least significant
                        ; digits of INT VAR.
MOV INT VAR+4, EDX
                        ; Store most significant
                         : digits of INT VAR.
```

; INT VAR now equals -8 (FFFFFFFF FFFFFF8H).

mgerIEX

Returns integer for number rounded to nearest with even preferred

# **Function**

ST := roundeven(ST)
Result := roundeven(x) as follows:

- Result := round\_to\_nearest(x) if fraction\_part(x) ≠ 0.5
- Result := (x + 0.5) if fraction part(x) = 0.5 and (x + 0.5) is even
- Result := (x 0.5) if fraction part(x) = 0.5 and (x + 0.5) is odd

#### Discussion

mqerIEX returns the nearest integer for the input x, where  $-\infty \le x \le +\infty$ . Results fall in the same range and remain in 80-bit extended format.

When x falls midway between two integers, mqerIEX rounds x to the even integer value. For example, mqerIEX returns:

- 3 for x = 3.3
- 4 for x = 4.5
- -6 for x = -6.5

For  $x = \pm \infty$ , mqerIEX returns x unchanged.

# **Exceptions**

Possible exceptions are Invalid, Denormal, and Precision:

- I Input SNaNs cause the I exception. If I is masked, mqerIEX returns the QNaN indefinite in ST. If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerIEX returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

## mgerIEX (continued)

P Input x values that are not integers cause the P exception. If P is masked, mgerIEX returns its result. If P is unmasked, control passes to the exception handler with the result in ST and the exception opcode set.

# **Exception Opcode**

178H

## Example

mgerIEX returns the nearest integer for an input number x in the range  $-\infty \le x \le +\infty$ ; its results fall in the same range.

```
: in DATA segment
                           ; initialized to test value
UNITS DO 5890.14
UNITS DQ 5890.14 ; initializ
ONE GRAND DD 1000.00 ; constant
THOUSANDS DT ?
                           ; in CODE32 segment
EXTRN MOERIEX: NEAR
: The following lines compute an integer number of
; thousands, rounding to the even THOUSANDS value
; if UNITS*ONE GRAND's fraction part equals 0.5.
FLD UNITS
                           ; push, ST := UNITS
                           ; ST := ST/1000
FDIV ONE GRAND
CALL MOERIEX
                         ; ST := roundeven(ST)
FSTP THOUSANDS
                           ; Store result, pop stack.
```

; THOUSANDS is now 6.00.

#### **Function**

ST := 
$$\log_{10}(ST)$$
  
Result :=  $\log_{10}(x)$  if  $0 < x \le +\infty$ 

#### Discussion

mqerLGD returns the result y such that  $10^y$  equals the input number x. For positive, nonzero x,  $\log_{10}(x)$  is a well-defined number. For example:

- $\log_{10}(1) = 0$
- $\log_{10}(100) = 2$
- $\log_{10}(50) \approx 1.69897$
- $\log_{10}(10) = 1$
- $\log_{10}(.001) = -3$

Finite results are in the range -4932...+4932. mqerLGD returns  $+\infty$  for  $x = +\infty$ .

## **Exceptions**

Possible exceptions are Invalid, Denormal, and Zerodivide:

- I Input  $-\infty \le x < 0$  and x = SNaN cause the I exception. If I is masked, mqerLGD returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerLGD returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.
- Z Input  $x = \pm 0$  causes the Z exception. If Z is masked, mqerLGD returns  $-\infty$ . If Z is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

# mqerLGD (continued) Exception Opcode

16DH

## Example

mqerLGD returns  $\log_{10}(x)$  where  $0 < x \le +\infty$ ; its finite results fall in the range -4932...+4932 or its result is  $+\infty$  for  $x = +\infty$ .

```
; : in DATA segment
QUANTITY DQ 0.0001 ; initialized to test value
TENS_POWER DQ ?

; : ; in CODE32 segment
EXTRN MQERLGD:NEAR

; The following lines return the common (base_10)
; logarithm of the input QUANTITY.

FLD QUANTITY ; push, ST := QUANTITY
CALL MQERLGD ; ST := log_base_10(ST)
FSTP TENS_POWER ; Store result, pop stack.

; The test input was 10 ** -4 but result could be
; approximately -4 because 0.0001 is not
```

; representable exactly in binary.

ST := ln(ST)Result := ln(x) where  $0 < x \le +\infty$  and  $ln(x) = log_e(x)$ 

#### **Function**

mqerLGE returns the result y such that  $e^y$  equals the input number x, where e is the transcendental constant 2.71828182845904523536... For positive, nonzero x, ln(x) is a well-defined number. For example:

- ln(1) = 0
- $ln(10) \approx 2.3025851$
- $ln(.001) \approx -6.9077533$

Finite results are in the range -11355..+11355. mqerLGE returns  $+\infty$  for  $x = +\infty$ .

# **Exceptions**

Possible exceptions are Invalid, Denormal, and Zerodivide:

- I Input  $-\infty \le x < 0$  and x = SNaN cause the I exception. If I is masked, mqerLGE returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerLGE returns its result. If D is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.
- Z Input  $x = \pm 0$  causes the Z exception. If Z is masked, mqerLGE returns  $-\infty$ . If Z is unmasked, control passes to the exception handler with x still in ST and the exception opcode set.

# **Exception Opcode**

16CH

# mqerLGE (continued) Example

mqerLGE returns ln(x) where  $0 < x \le +\infty$ ; its finite results fall in the range -11355..+11355 or its result is  $+\infty$  for  $x = +\infty$ .

```
: in DATA segment
                       ; initialized to test value
X DO -3.0
ONE DD 1.00
                        ; constant
THETA DO ?
                        : in CODE32 segment
EXTRN MOERLGE: NEAR
The following lines return a radian angle THETA
; as an inverse hyperbolic sine according to
: the standard mathematical formula:
: arcsinh(x) = ln(x + square root(x**2 + 1)).
FLD X
                         : push. ST := X
FMUL ST,ST
                         : ST := ST*ST
FADD ONE
                        ; ST := ST + 1
FSQRT
                         : ST := square root(ST)
                        : ST := ST + X
FADD X
CALL MQERLGE
                         : ST := ln(ST)
FSTP THETA
                         : Store result, pop stack.
: THETA is now about -1.8184465.
```

# Function

ST := maximum(ST(1),ST)
Result := maximum(x,y) as follows:

- Result := x if x > y
- Result := y if  $x \le y$

#### Discussion

mqerMAX returns the greater of x and y, where x and y are numbers in the range  $-\infty..+\infty$ .

Results are in the same range. If x = y, mqerMAX returns y (last-pushed argument), even when  $x = y = \pm 0$ .

# **Exceptions**

Possible exceptions are Invalid and Denormal:

- I Input SNaNs and QNaNs cause the I exception. mqerMAX relies implicitly on the trichotomy law to determine its result, and NaNs are unordered with respect to the representable numbers. If I is masked, mqerMAX returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST(1), y still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerMAX returns its result. If D is unmasked, control passes to the exception handler with x still in ST(1), y still in ST, and the exception opcode set.

# **Exception Opcode**

282H

# mqerMAX (continued) Example

mqerMAX returns the greater of (x,y), where x and y are numbers in the range  $-\infty..+\infty$ ; its results fall in the same range with a result of y (last-pushed argument) if x = y.

```
: in DATA segment
                        ; initialized
X VAR1 DO -5.3
                        : to test values
Y VAR2 DO -7
LARGER DO
           ?
                         : in CODE32 segment
EXTRN MOERMAX: NEAR
: The following lines set LARGER to the maximum
; of X VAR1 and Y VAR2.
FLD X VAR1
                          : push, ST := X VAR1
                          : push, ST := Y VAR2
FLD Y VAR2
CALL MQERMAX
                         ; ST := max(ST(\overline{1}),ST)
                        : Store result, pop stack.
FSTP LARGER
: LARGER is now -5.3.
```

ST := minimum(ST(1),ST)
Result := minimum(x,y) as follows:

- Result := x if x < y
- Result := y if  $x \ge y$

#### **Function**

mqerMIN returns the lesser of x and y, where x and y are numbers in the range  $-\infty..+\infty$ .

Results are in the same range. If x = y, mqerMIN returns y (last-pushed argument), even when  $x = y = \pm 0$ .

# **Exceptions**

Possible exceptions are Invalid and Denormal:

- I Input SNaNs and QNaNs cause the I exception. mqerMIN relies implicitly on the trichotomy law to determine its result, and NaNs are unordered with respect to the representable numbers. If I is masked, mqerMIN returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST(1), y still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerMIN returns its result. If D is unmasked, control passes to the exception handler with x still in ST(1), y still in ST, and the exception opcode set.

### **Exception Opcode**

281H

# mqerMIN (continued) Example

mqerMIN returns the lesser of (x,y), where x and y are numbers in the range  $-\infty..+\infty$ ; its results fall in the same range with a result of y (last-pushed argument) if x = y.

```
; in DATA seament
        .
X VAR1 DO -5.3
                       : initialized
Y VAR2 DO -7.9
                      : to test values
SMALLER DO ?
                         : in CODE32 segment
EXTRN MOERMIN: NEAR
: The following lines set SMALLER to the minimum
; of X VAR1 and Y VAR2.
FLD X VAR1
                        ; push, ST := X VAR1
FLD Y VAR2
                         : push, ST := Y VAR2
CALL MQERMIN
                        ; ST := minimum(ST(1).ST)
FSTP SMALLER
                         : Store result, pop stack.
: SMALLER is now -7.9.
```

# **Function**

 $ST := (ST(1) \mod ST)$  with sign of ST(1)Result := x mod y with sign of x, as follows:

- Result := x (y \* chop(x/y)) if  $-\infty < x < +\infty$  and  $y \ne 0$
- $chop(x/y) = (x/y) (fraction_part(| x/y |) with sign of x/y)$

#### Discussion

mqerMOD returns the modulus of x and y for  $-\infty < x < +\infty$  and  $-\infty \le y < 0$  or  $0 < y \le +\infty$ . mqerMOD extends the range for x and y beyond the 80387 FPREM instruction's. The result's absolute value is always less than | y |, signed with the same sign as x.

mqerMOD subtracts an integer multiple of y from the input x to bring x down to within y units of zero. For example, if  $y = \pm 5$ , mqerMOD returns a result between, but not including, -5 and +5 as follows:

- mqerMOD(-7.5) = -2
- mqerMOD (-10,5) = -0
- mqerMOD (-19.99, -5) = -4.99
- mqerMOD (19.99, -5) = 4.99
- mqerMOD (45,5) = +0
- mqerMOD (44.75,5) = 4.75

If  $y = \pm \infty$ , mqerMOD returns x unchanged as long as x is a finite, representable number or a QNaN.

mqerMOD places the three least significant bits of chop(x/y) in the condition code bits  $C_3C_1C_0$  of the 80387 Status Word.  $C_0$  has the least significant bit,  $C_1$  has the next bit, and  $C_3$  has the most significant bit of these chop(x/y) bits.

# mqerMOD (continued) Exceptions

Possible exceptions are Invalid and Denormal:

- I Input  $x = \pm \infty$ ,  $y = \pm 0$ , and (x or y) = SNaN cause the I exception. If I is masked, mqerMOD returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST(1), y still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerMOD returns its result. If D is unmasked, control passes to the exception handler with x still in ST(1), y still in ST, and the exception opcode set.

## **Exception Opcode**

269H

### Example

mqerMOD returns (x - (y \* chop(x/y))) for x in the range  $-\infty < x < +\infty$  and y in the range  $-\infty \le y < 0$  or  $0 < y \le +\infty$ . Its results fall in the ranges  $-y < (x \mod y) \le 0$  for negative x and  $0 \le (x \mod y) < y$  for positive x.

```
: : ; in DATA segment
X DQ 181137.00 ; initialized to test value
ONE_GRAND DD 1000.00 ; constant
LAST_THREE_DIGITS DQ ?
```

```
: ; in CODE32 segment EXTRN MOERMOD: NEAR
```

; The following lines calculate X MOD 1000.

```
FLD X ; push, ST := X
FLD ONE_GRAND ; push, ST := ONE_GRAND
CALL MQERMOD ; ST := ST(1) mod ST
FSTP LAST_THREE_DIGITS ; Store result, pop stack.
```

```
; LAST_THREE_DIGITS is 137.00.
```

# **Function**

 $ST := (ST(1) \mod ST)$  with sign of ST(1)Result := x mod y with sign of x, as follows:

- Result :=  $x (y * \operatorname{chop}(x/y))$  if  $-\infty < x < +\infty$  and  $y \neq 0$
- chop(x/y) = (x/y) (fraction part(| x/y |) with sign of x/y)

#### Discussion

mqerMOD returns the modulus of x and y for  $-\infty < x < +\infty$  and  $-\infty \le y < 0$  or  $0 < y \le +\infty$ . mqerMOD extends the range for x and y beyond the 80387 FPREM instruction's. The result's absolute value is always less than |y|, signed with the same sign as x.

mqerMOD subtracts an integer multiple of y from the input x to bring x down to within y units of zero. For example, if  $y = \pm 5$ , mqerMOD returns a result between, but not including, -5 and +5 as follows:

- mqerMOD(-7.5) = -2
- mqerMOD (-10,5) = -0
- mqerMOD (-19.99, -5) = -4.99
- mqerMOD (19.99, -5) = 4.99
- mqerMOD (45,5) = +0
- mqerMOD (44.75,5) = 4.75

If  $y = \pm \infty$ , mqerMOD returns x unchanged as long as x is a finite, representable number or a QNaN.

mqerMOD places the three least significant bits of chop(x/y) in the condition code bits  $C_3C_1C_0$  of the 80387 Status Word.  $C_0$  has the least significant bit,  $C_1$  has the next bit, and  $C_3$  has the most significant bit of these chop(x/y) bits.

# mqerMOD (continued) Exceptions

Possible exceptions are Invalid and Denormal:

- I Input  $x = \pm \infty$ ,  $y = \pm 0$ , and (x or y) = SNaN cause the I exception. If I is masked, mqerMOD returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input x still in ST(1), y still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerMOD returns its result. If D is unmasked, control passes to the exception handler with x still in ST(1), y still in ST, and the exception opcode set.

## **Exception Opcode**

269H

#### Example

mqerMOD returns (x - (y \* chop(x/y))) for x in the range  $-\infty < x < +\infty$  and y in the range  $-\infty \le y < 0$  or  $0 < y \le +\infty$ . Its results fall in the ranges  $-y < (x \mod y) \le 0$  for negative x and  $0 \le (x \mod y) < y$  for positive x.

```
: : ; in DATA segment
X DQ 181137.00 ; initialized to test value
ONE GRAND DD 1000.00 ; constant
LAST_THREE_DIGITS DQ ?

: : ; in CODE32 segment
EXTRN MQERMOD: NEAR

; The following lines calculate X MOD 1000.

FLD X ; push, ST := X
FLD ONE GRAND ; push, ST := ONE GRAND
CALL MQERMOD ; ST := ST(1) mod ST
```

FSTP LAST THREE DIGITS ; Store result, pop stack.

; LAST\_THREE\_DIGITS is 137.00.

#### **Function**

ST := (ST(1) rem ST) Result := x rem y as follows:

- Result := x (y \* roundeven(x/y)) if  $x \neq \pm \infty$  and  $y \neq 0$
- roundeven(x/y) =
  - round\_to\_nearest(x/y) if fraction\_part(x/y) ≠ 0.5
  - (x/y + 0.5) if fraction part(x/y) = 0.5 and (x/y + 0.5) is even
  - (x/y 0.5) if fraction part(x/y) = 0.5 and (x/y + 0.5) is odd

#### Discussion

mqerRMD returns the remainder of x and y where  $-\infty < x < +\infty$  and  $-\infty \le y < 0$  or  $0 < y \le +\infty$ . mqerRMD extends the range for x and y beyond the 80387 FPREM1 instruction's. Results are in the range -y/2..+y/2.

mqerRMD subtracts an integer multiple of y from the input x to bring x down to within y/2 units of zero. For example, if  $y = \pm 5$ , mqerMOD returns a result between -2.5 and +2.5 as follows:

- mqerRMD(-7,5) = -2
- mqerRMD(-10.5) = 0
- mqerRMD(-19.99, -5) = +0.01
- mqerRMD (19.99, -5) = -0.01
- mgerRMD(44.75,5) = -0.25
- mqerRMD(2.5,5) = 2.5
- mqerRMD (7.5,5) = -2.5

For x that are odd integer multiples of y/2, results alternate between -y/2 and +y/2. mqerRMD returns +y/2 for input x values in the series {... -7y/2, -3y/2, y/2, 5y/2, 9y/2, ...}. It returns -y/2 for input x values in the series {... -5y/2, -y/2, 3y/2, 7y/2, 11y/2, ...}.

## mgerRMD (continued)

If  $y = \pm \infty$ , mqerRMD returns x unchanged as long as x is a finite, representable number or a QNaN.

mqerRMD places the three least significant bits of roundeven(x/y) in the condition code bits  $C_3C_1C_0$  of the 80387 Status Word.  $C_0$  has the least significant bit,  $C_1$  has the next bit, and  $C_3$  has the most significant bit of these roundeven(x/y) bits.

## **Exceptions**

Possible exceptions are Invalid and Denormal:

- I Input  $x = \pm \infty$ ,  $y = \pm 0$ , and (x or y) = SNaN cause the I exception. If I is masked, mqerRMD returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with x still in ST(1), y still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerRMD returns its result. If D is unmasked, control passes to the exception handler with x still in ST(1), y still in ST, and the exception opcode set.

# **Exception Opcode**

27AH

## Example

mqerRMD returns (x - (y \* roundeven(x/y))) where  $-\infty < x < +\infty$  and  $y \ne 0$ ; its results are in the range -y/2..+y/2.

```
: ; in DATA segment
```

EXTRN MQERRMD: NEAR

```
; The following lines reduce the angle THETA RADIANS
```

; to its principal value.

```
FLD THETA_RADIANS ; push, ST := THETA_RADIANS
```

; THETA RADIANS is now about 0.2831853.

CALL MQERRMD ; ST := ST(1) rem ST

FSTP THETA\_RADIANS ; Store result, pop stack.

# mgerSGN

Returns number with other number's sign

#### Function

ST := ST(1) with sign of ST Result := x with sign of y as follows:

- Result := |x| if  $y \ge 0$
- Result := -|x| if y < 0

#### Discussion

mqerSGN returns the absolute value of x with the sign of y, where x and y are numbers in the range  $-\infty..+\infty$ . Results are in the same range.

Along with  $\pm \infty$  for x, mqerSGN accepts any bit pattern in extended format (80 bits) without signaling an exception. However, y must be a signed number or  $\pm \infty$ . mqerSGN returns + |x| if y = -0.

# **Exceptions**

Possible exceptions are Invalid and Denormal:

- I Input (x or y) = SNaN or y = QNaN causes the I exception.
  mqerSGN relies implicitly on the trichotomy law to determine the
  sign of its result, and NaNs are unordered with respect to the
  representable numbers. If I is masked, mqerSGN returns the
  QNaN indefinite. If I is unmasked, control passes to the
  exception handler with the input x still in ST(1), y still in ST, and
  the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerSGN returns its result. If D is unmasked, control passes to the exception handler with x still in ST(1), y still in ST, and the exception opcode set.

# **Exception Opcode**

264H

## Example

mqerSGN returns | x | with the sign of y, where x and y are numbers in the range  $-\infty..+\infty$ ; its results fall in the same range.

```
; in DATA segment
Y COOR DQ -0.001
                         : initialized to test value
THETA RADIANS DQ ?
                         ; in CODE32 segment
EXTRN MQERSGN: NEAR
; The following lines return the angle pi
; with the sign of Y COOR.
FLDPI
                         ; push, ST := pi
                         ; push, ST := Y COOR
FLD Y COOR
CALL MQERSGN
                        ; ST := ST(1) with sign of ST
FSTP THETA RADIANS
                         ; Store result, pop stack.
; THETA RADIANS is now -pi.
```

# mgerSIN

Returns trigonometric sine

#### **Function**

```
ST := \sin(ST)
Result := \sin(\theta) if -\infty < \theta < +\infty
```

#### Discussion

mqerSIN returns the trigonometric sine of an angle  $\theta$ , where  $-\infty < \theta < +\infty$  and  $\theta$  is expressed in radians. mqerSIN extends the range for  $\theta$  beyond the 80387 FSIN instruction's.

Results are in the range -1.0..+1.0. For  $\theta = \pm 0$ , mqerSIN returns  $\theta$  unchanged.

# **Exceptions**

Possible exceptions are Invalid and Denormal:

- I Input  $\theta = \pm \infty$  and  $\theta = \text{SNaN}$  cause the I exception. If I is masked, mqerSIN returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input  $\theta$  still in ST and the exception opcode set.
- **D** Input denormals cause the D exception. If D is masked, mqerSIN returns its result. If D is unmasked, control passes to the exception handler with  $\theta$  still in ST and the exception opcode set.

## **Exception Opcode**

171H

### Example

mqerSIN returns  $sin(\theta)$  for  $\theta$  a finite angle in radians; its results fall in the range -1.0..+1.0.

```
; in DATA segment
POLAR THETA DQ 30.0
                       : initialized
POLAR RADIUS DQ 2.0 to test values
DEG TO RAD DT 3FF98EFA351294E9C8AER
                         : constant pi/180
REC Y DQ ?
    : :
                         ; in CODE32 segment
EXTRN MQERSIN: NEAR
: The following lines compute the Y-coordinate
; of a polar-to-rectangular conversion.
FLD POLAR THETA
                         ; push, ST := POLAR THETA
FLD DEG TO RAD
                         : push, ST := DEG TO RAD
                        ; ST(1) := ST(1)*\overline{ST}, pop ST
FMUL
                        ; ST := sin(ST)
CALL MQERSIN
                        ; ST := ST*POLAR RADIUS
FMUL POLAR RADIUS
FSTP REC Y
                         : Store result, pop stack.
; REC Y is now 1.0.
```

# mgerSNH

Returns hyperbolic sine

#### **Function**

```
ST := \sinh(ST)
Result := \sinh(\theta) if -11355.13 \le \theta \le +11355.13 or \theta = \pm \infty
```

#### Discussion

mqerSNH returns the hyperbolic sine of an angle  $\theta$ , where -11355.13  $\leq \theta \leq +11355.13$  or  $\theta = \pm \infty$ .

Results are in the range  $-\infty..+\infty$ . For  $\theta = \pm 0$  or  $\theta = \pm \infty$ , mqerSNH returns  $\theta$  unchanged.

# **Exceptions**

Possible exceptions are Invalid, Denormal, and Overflow:

- I Input SNaNs cause the I exception. If I is masked, mqerSNH returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input  $\theta$  still in ST and the exception opcode set.
- **D** Input denormals cause the D exception. If D is masked, mqerSNH returns its result. If D is unmasked, control passes to the exception handler with  $\theta$  still in ST and the exception opcode set.
- O Input  $\theta$  outside the range -11355.13..+11355.13 but not equal to  $\pm \infty$  cause the O exception. If O is masked, mqerSHN returns  $-\infty$  for  $\theta$  < -11355.13 and  $+\infty$  for  $\theta$  > +11355.13. If O is unmasked, control passes to the exception handler with  $\theta$  still in ST and the exception opcode set.

# **Exception Opcode**

16EH

## Example

mqerSNH returns  $sinh(\theta)$  where  $-11355.13 \le \theta \le +11355.13$  or  $\theta = \pm \infty$ ; its results fall in the range  $-\infty..+\infty$ .

: ; in DATA segment
THETA DQ -2.7 ; initialized to test value

SINH\_VALUE DQ ?

: ; in CODE32 segment

EXTRN MQERSNH: NEAR

; The following lines compute sinh(THETA)

; and store the result in SINH\_VALUE.

FLD THETA ; push, ST := THETA CALL MQERSNH ; ST := sinh(ST)

FSTP SINH\_VALUE ; Store result, pop stack.

; SINH\_VALUE is now about -7.4062628.

# mgerTAN

Returns trigonometric tangent

#### **Function**

ST := 
$$tan(ST)$$
  
Result :=  $tan(\theta)$  if  $-\infty < \theta < +\infty$ 

#### Discussion

mqerTAN returns the trigonometric tangent of an angle  $\theta$ , where  $-\infty < \theta < +\infty$  and  $\theta$  is expressed in radians. mqerTAN extends the range for  $\theta$  beyond the 80387 FPTAN instruction's.

Results are in the range  $-\infty..+\infty$ . For  $\theta = \pm 0$ , mqerTAN returns  $\theta$  unchanged.

# **Exceptions**

Possible exceptions are Invalid and Denormal:

- I Input  $\theta = \pm \infty$  and  $\theta = \text{SNaN}$  cause the I exception. If I is masked, mqerTAN returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input  $\theta$  still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerTAN returns its result. If D is unmasked, control passes to the exception handler with  $\theta$  still in ST and the exception opcode set.

# **Exception Opcode**

173H

### Example

mqerTAN returns  $tan(\theta)$  where  $-\infty < \theta < +\infty$  and  $\theta$  is an angle in radians; its results fall in the range  $-\infty..+\infty$ .

: ; in DATA segment

THETA DEGREES DQ 45.00 ; initialized to test value

DEG TO RAD DT 3FF98EFA351294E9C8AER

; constant pi/180

SLOPE DO ?

: ; in CODE32 segment

EXTRN MQERTAN: NEAR

; The following lines compute tan(THETA DEGREES)

; to determine the slope of a line in the x-y plane.

FLD THETA DEGREES ; push, ST := THETA DEGREES

FLD DEG TO RAD ; push, ST :=  $pi/18\overline{0}$ 

FMUL ; ST(1) := ST(1)\*ST, pop ST

CALL MQERTAN ; ST := tan(ST)

FSTP SLOPE ; Store result, pop stack.

; SLOPE is now 1.0.

# mgerTNH

Returns hyperbolic tangent

#### **Function**

```
ST := tanh(ST)
Result := tanh(\theta) if -\infty \le \theta \le +\infty
```

#### Discussion

mqerTNH returns the hyperbolic tangent of an angle  $\theta$ , where  $-\infty \le \theta \le +\infty$ .

Results are in the range -1.0..+1.0. mqerTNH returns -1.0 for  $-\infty \le \theta$  < -32 and +1.0 for +32 <  $\theta \le +\infty$ . For  $\theta = \pm 0$ , mqerTNH returns  $\theta$  unchanged.

# **Exceptions**

Possible exceptions are Invalid and Denormal:

- I Input SNaNs cause the I exception. If I is masked, mqerTNH returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input  $\theta$  still in ST and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerTNH returns its result. If D is unmasked, control passes to the exception handler with  $\theta$  still in ST and the exception opcode set.

## **Exception Opcode**

170H

## Example

mgerTNH returns  $tanh(\theta)$ ; its results fall in the range -1.0..+1.0.

: ; in DATA segment

THETA DQ -0.62 ; initialized to test value

TANH\_VALUE DQ ?

: : ; in CODE32 segment

EXTRN MOERTNH: NEAR

: The following lines compute tanh(THETA)

; and store the result in TANH VALUE.

FLD THETA ; push, ST := THETA CALL MQERTNH ; ST := tanh(ST)

FSTP TANH VALUE Store result, pop stack.

; TANH VALUE is now about -0.5511281.

# mgerY2X

Returns real number raised to a real or integer power

#### **Function**

ST :=  $ST(1)^{ST}$ Result :=  $y^x$  or Result :=  $y^j$  where x is a non-integer and j is an integer as follows:

- Result := y if y =  $\pm 0$  and (x or j) > 0
- Result :=  $y^x$  if  $0 < y < +\infty$  and  $-\infty < x < +\infty$
- Result := 1.0 if  $y \neq \pm \infty$  and  $j = \pm 0$
- Result :=  $y^j$  if  $-\infty < y < +\infty$  and  $j \ne \pm 0$  as follows:
  - Result :=  $y^j$  if  $0 < y < +\infty$  and j > 0
  - Result :=  $|y|^{j}$  if  $-\infty < y < 0$  and j is even
  - Result :=  $-(|y|^j)$  if  $-\infty < y < 0$  and j is odd

#### Discussion

mqerY2X returns  $y^x$  or  $y^j$  where y is a number, x is a non-integer, and j is an integer; y, x, and j are in 80-bit extended format as are the results

For finite y > 0 with a finite, non-integer exponent, mqerY2X returns  $2^{(x*\log_2 base_2(y))}$  for  $y^x$ . mqerY2X accepts  $y = +\infty$ ,  $y = \pm 0$ , or  $x = \pm \infty$  under certain conditions and returns:

- $+\infty$  if  $y = +\infty$  and x > 0
- +0 if  $y = +\infty$  and  $-\infty \le x < 0$
- y if  $y = \pm 0$  and  $0 < x \le +\infty$
- +0 if  $\pm 0 < y < +1.0$  and  $x = +\infty$
- $+\infty$  if  $+1.0 < y \le +\infty$  and  $x = +\infty$
- $+\infty$  if  $\pm 0 \le y < +1.0$  and  $x = -\infty$
- +0 if +1.0 < y  $\leq$  + $\infty$  and x = - $\infty$

# mqerY2X (continued)

For finite y > 0 with an integer exponent, mqerY2X returns:

- $2^{(j*log\_base\_2(\gamma))}$  if |i| > 63
- $y^j$  if  $1 \le j \le 63$  by successively squaring and multiplying y to compute the correct power
- $y^j$  if  $-63 \le j \le -1$ , by evaluating one of two possible expressions:
  - 1 / (y | j |) provided that the denominator would not cause numeric overflow or underflow
  - (1/y) | j | otherwise

For finite y < 0 with an integer exponent, mqerY2X first computes  $|y|^j$  as for finite y > 0. Then, it determines the sign of the result according to whether j is even (+) or odd (-).

For  $|j| \le 63$ , mqerY2X performs no more than nine multiplications in evaluating (y\*y \*(y\*y) ...\*(y\*...\*y)) with its squaring-and-multiplying algorithm.

mqerY2X accepts  $y = \pm \infty$  or  $y = \pm 0$  with an integer exponent under certain conditions and returns:

- $+\infty$  if  $y = +\infty$  and i > 0
- +0 if  $y = +\infty$  and j < 0
- $+\infty$  if  $y = -\infty$ , i > 0, and j is even
- $-\infty$  if  $y = -\infty$ , j > 0, and j is odd
- +0 if  $y = -\infty$ , j < 0, and j is even
- -0 if  $y = -\infty$ , j < 0, and j is odd
- $y \text{ if } y = \pm 0 \text{ and } j > 0$

# mqerY2X (continued) Exceptions

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

- I Input SNaNs, y = ±∞ with a zero exponent, -∞ ≤ y < 0 with an infinite or non-integer exponent, and y = +1.0 with an infinite exponent cause the I exception. If I is masked, mqerY2X returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input y still in ST(1), the input exponent still in ST, and the exception opcode set.</p>
- D Input denormals cause the D exception. If D is masked, mqerY2X returns its result. If D is unmasked, control passes to the exception handler with y still in ST(1), the exponent still in ST, and the exception opcode set.
- Z Input y = ±0 with a finite (x or j) < 0 causes the Z exception. If Z is masked, mqerY2X returns ∞ with the sign of y. If Z is unmasked, control passes to the exception handler with y still in ST(1), the exponent still in ST, and the exception opcode set.
- O A y<sup>x</sup> or y<sup>j</sup> whose exponent is too large to be represented in the extended format causes the O exception. If O is masked, mqerY2X returns ∞ with the same sign as the true result. If O is unmasked, control passes to the exception handler with the input y in ST(1), the input exponent in ST, and the exception opcode set.
- U A |  $y^x$  | or |  $y^j$  | too tiny to fit accurately in extended format causes the U exception if U is masked. In this case, mqerY2X returns a gradual underflow denormal, if possible; otherwise, it returns  $\pm 0$ . A |  $y^x$  | or |  $y^j$  | less than the smallest positive normalized number causes the U exception if U is unmasked. In this case, control passes to the exception handler with the input y in ST(1), the exponent in ST, and the exception opcode set.

## **Exception Opcode**

26AH

## **Example**

mqerY2X returns yx or yj.

```
: : ; in DATA segment INPUT_VALUE DQ 64.00 ; initialized to test value
CUBE ROOT DQ ?
                       : in CODE32 segment
EXTRN MQERY2X: NEAR
; The following lines take a cube root
; and store the result in CUBE ROOT.
FLD INPUT VALUE
                      ; push, ST := INPUT VALUE
FLD ONE THIRD
                      ; push, ST := ONE THIRD
                      ; ST := ST(1)**ST
CALL MOERY2X
FSTP CUBE_ROOT
                      : Store result, pop stack.
; CUBE ROOT is now about 4.0.
```

# mqerYl2

Returns real number raised to a word integer power

#### **Function**

ST := ST<sup>AX</sup> Result :=  $y^j$  if  $-\infty \le y \le +\infty$  and  $-32,768 \le j \le +32,767$  as follows:

- Result := 1.0 if j = 0
- Result :=  $\pm 0$  if  $y = \pm 0$ , j > 0, and j is even
- Result := y if  $y = \pm 0$ , j > 0, and j is odd
- Result :=  $y^j$  if  $0 < y < +\infty$  and  $-32,768 \le j < 0$  or  $0 < j \le +32,767$
- Result :=  $|y|^j$  if  $-\infty < y < 0$ ,  $j \ne 0$ , and j is even
- Result :=  $-(|y|^j)$  if  $-\infty < y < 0$  and j is odd

#### Discussion

mqerYI2 returns  $y^j$ , where y is a real number and j is a 16-bit integer, expressed in two's complement. If j = 0, mqerYI2 returns +1.0, whatever the value of y.

For finite y > 0, mqerYI2 returns:

- $2^{(j*log\_base\_2(y))}$  if |j| > 63
- $y^j$  if  $1 \le j \le 63$  by successively squaring and multiplying y to compute the correct power
- $y^j$  if  $-63 \le j \le -1$  by evaluating one of two possible expressions:
  - 1 / (y | j |) provided that the denominator would not cause numeric overflow or underflow
  - (1/y) | j | otherwise

For finite y < 0, mqerYI2 first computes  $|y|^j$  as for finite y > 0. Then, it determines the sign of the result according to whether j is even (+) or odd (-). For  $|j| \le 63$ , mqerYI2 performs no more than nine multiplications in evaluating (y\*y\*(y\*y) ...\*(y\*...\*y)) with its squaring-and-multiplying algorithm.

mqerYI2 accepts input  $y = \pm \infty$  or  $y = \pm 0$  and returns:

- $+\infty$  if  $y = +\infty$  and i > 0
- +0 if  $y = +\infty$  and i < 0
- $+\infty$  if  $y = -\infty$ , j > 0, and j is even
- $-\infty$  if  $y = -\infty$ , j > 0, and j is odd
- +0 if  $y = -\infty$ , j < 0, and j is even
- -0 if  $y = -\infty$ , j < 0, and j is odd
- +0 if  $y = \pm 0$ , j > 0, and j is even
- y if  $y = \pm 0$ , j > 0, and j is odd

# **Exceptions**

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

- I Input y = SNaN causes the I exception. If I is masked, mqerYI2 returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerYI2 returns its result. If D is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- Z Input  $y = \pm 0$  with j < 0 causes the Z exception. If Z is masked, mqerYI2 returns  $\infty$  with the same sign as y. If Z is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- O A y<sup>j</sup> whose exponent is too large to be represented in the extended format causes the O exception. If O is masked, mqerYI2 returns ∞ with the same sign as the true result. If O is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.

#### mgerYI2 (continued)

U A | y<sup>j</sup> | too tiny to fit accurately in extended format causes the U exception if U is masked. In this case, mqerYI2 returns a gradual underflow denormal, if possible; otherwise, it returns ±0. A | y<sup>j</sup> | less than the smallest possible normalized number causes the U exception if U is unmasked. In this case, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.

## **Exception Opcode**

27CH

## Example

mqerYI2 raises a real number y on the 80387 stack to the power of a 16-bit integer in the AX register. Its results fall in the range of finite, representable numbers or its results are ±\infty.

```
: : ; in DATA segment
INT_POWER DW 9 : ; initialized
REAL_Y DQ 1.3 ; to test values
REAL_OUTPUT DQ ?

: : ; in CODE32 segment
EXTRN MQERYI2: NEAR

; The following lines raise REAL_Y to INT_POWER
; and store the result in REAL_OUTPUT.

FLD REAL_Y
MOV AX,INT_POWER
CALL MQERYI2 ; ST := ST**AX
FSTP REAL_OUTPUT ; Store result, pop ST.

; REAL_OUTPUT is now about 10.6045.
```

Returns real number raised to a short integer power

# Function

ST := ST<sup>EAX</sup>
Result :=  $y^j$  if  $-\infty \le y \le +\infty$  and  $-2,147,483,648 \le j \le +2,147,483,647$   $(-2^{31} \le j \le +(2^{31}-1))$  as follows:

- Result := 1.0 if i = 0
- Result :=  $\pm 0$  if  $y = \pm 0$ , j > 0, and j is even
- Result := y if  $y = \pm 0$ , j > 0, and j is odd
- Result :=  $y^j$  if  $0 < y < +\infty$  and  $-2^{31} \le j < 0$  or  $0 < j \le +(2^{31} 1)$
- Result :=  $|y|^j$  if  $-\infty < y < 0$ ,  $j \ne 0$ , and j is even
- Result :=  $-(|y|^j)$  if  $-\infty < y < 0$  and j is odd

#### Discussion

mqerYI4 returns  $y^{j}$ , where y is a real number and j is a 32-bit integer, expressed in two's complement. If j = 0, mqerYI4 returns +1.0, whatever the value of y.

For finite y > 0, mgerYI4 returns:

- $2^{(j*log\_base\_2(y))}$  if |j| > 63
- $y^j$  if  $1 \le j \le 63$  by successively squaring and multiplying y to compute the correct power
- $y^j$  if  $-63 \le j \le -1$  by evaluating one of two possible expressions:
  - 1 / (y j j ) provided that the denominator would not cause numeric overflow or underflow
  - (1/y) otherwise

For finite y < 0, mqerYI4 first computes  $|y|^j$  as for finite y > 0. Then, it determines the sign of the result according to whether j is even (+) or odd (-).

# mgerYI4 (continued)

For  $|j| \le 63$ , mqerYI4 performs no more than nine multiplications in evaluating  $(y^*y^*(y^*y) ...^*(y^*...^*y))$  with its squaring-and-multiplying algorithm.

mqerYI4 accepts input  $y = \pm \infty$  or  $y = \pm 0$  and returns:

- $+\infty$  if  $y = +\infty$  and j > 0
- +0 if  $v = +\infty$  and i < 0
- $+\infty$  if  $y = -\infty$ , j > 0, and j is even
- $-\infty$  if  $y = -\infty$ , j > 0, and j is odd
- +0 if  $y = -\infty$ , j < 0, and j is even
- -0 if  $y = -\infty$ , j < 0, and j is odd
- +0 if  $y = \pm 0$ , j > 0, and j is even
- y if  $y = \pm 0$ , j > 0, and j is odd

## **Exceptions**

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

- I Input y = SNaN causes the I exception. If I is masked, mqerYI4 returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerYI4 returns its result. If D is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- Z Input  $y = \pm 0$  with j < 0 causes the Z exception. If Z is masked, mqerYI4 returns  $\infty$  with the same sign as y. If Z is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- O A y<sup>j</sup> whose exponent is too large to be represented in the extended format causes the O exception. If O is masked, mqerYI4 returns  $\infty$  with the same sign as the true result. If O is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.

## mqerYI4 (continued)

U A | y<sup>j</sup> | too tiny to fit accurately in extended format causes the U exception if U is masked. In this case, mqerYI4 returns a gradual underflow denormal, if possible; otherwise, it returns ±0. A | y<sup>j</sup> | less than the smallest positive normalized number causes the U exception if U is unmasked. In this case, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.

### **Exception Opcode**

27CH

### Example

mqerYI4 raises a real number y on the 80387 stack to the power of a 32-bit integer in the EAX register. Its results fall in the range of finite, representable numbers or its results are  $\pm \infty$ .

```
: in DATA segment
INT POWER DD 9
                         : initialized
REAL Y DQ 2.3
                         : to test values
REAL OUTPUT DQ ?
                         ; in CODE32 segment
EXTRN MOERYI4: NEAR
; The following lines raise REAL Y to INT POWER
; and store the result in REAL OUTPUT.
FLD REAL Y
                         : push, ST := REAL Y
MOV EAX, INT POWER
                         : EAX := INT POWER
                        : ST := ST**EAX
CALL MOERYI4
FSTP REAL OUTPUT
                         ; Store result, pop ST.
; REAL OUTPUT is now about 1801.15.
```

# mgerY18

Returns real number raised to a long integer power

#### **Function**

 $ST := ST^{EDX}_{EAX}$ 

Result :=  $y^j$  if  $-\infty \le y \le +\infty$  and  $-2^{63} \le j \le +(2^{63} - 1)$  as follows:

- Result := 1.0 if j = 0
- Result :=  $\pm 0$  if  $y = \pm 0$ , j > 0, and j is even
- Result := y if  $y = \pm 0$ , j > 0, and j is odd
- Result :=  $y^j$  if  $0 < y < +\infty$  and  $-2^{63} \le j < 0$  or  $0 < j \le +(2^{63} 1)$
- Result :=  $|y|^j$  if  $-\infty < y < 0$ ,  $j \ne 0$ , and j is even
- Result :=  $-(|y|^j)$  if  $-\infty < y < 0$  and j is odd

#### Discussion

mqerYI8 returns  $y^j$ , where y is a real number and j is a 64-bit integer, expressed in two's complement. If j = 0, mqerYI8 returns +1.0, whatever the value of y.

For finite y > 0, mqerYI8 returns:

- $2^{(j*log\_base\_2(y))}$  if |j| > 63
- $y^j$  if  $1 \le j \le 63$  by successively squaring and multiplying y to compute the correct power
- $y^j$  if  $-63 \le j \le -1$  by evaluating one of two possible expressions:
  - 1 / (y | j |) provided that the denominator would not cause numeric overflow or underflow
  - (1/y) otherwise

For finite y < 0, mqerYI8 first computes  $|y|^j$  as for finite y > 0. Then, it determines the sign of the result according to whether j is even (+) or odd (-). For  $|j| \le 63$ , mqerYI8 performs no more than nine multiplications in evaluating  $(y^*y^*(y^*y) ...^*(y^*...^*y))$  with its squaring-and-multiplying algorithm.

mqerYI8 accepts input  $y = \pm \infty$  or  $y = \pm 0$  and returns:

- $+\infty$  if  $y = +\infty$  and j > 0
- +0 if  $y = +\infty$  and i < 0
- $+\infty$  if  $y = -\infty$ , j > 0, and j is even
- $-\infty$  if  $y = -\infty$ , j > 0, and j is odd
- +0 if  $y = -\infty$ , j < 0, and j is even
- -0 if  $y = -\infty$ , j < 0, and j is odd
- $\pm 0$  if  $y = \pm 0$ , i > 0, and i is even
- y if  $y = \pm 0$ , i > 0, and i is odd

## **Exceptions**

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

- I Input y = SNaN causes the I exception. If I is masked, mqerYI8 returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerY18 returns its result. If D is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- Z Input  $y = \pm 0$  with j < 0 causes the Z exception. If Z is masked, mqerYI8 returns  $\infty$  with the same sign as y. If Z is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- O A y<sup>j</sup> whose exponent is too large to be represented in the extended format causes the O exception. If O is masked, mqerY18 returns  $\infty$  with the same sign as the true result. If O is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.

### mgerYI8 (continued)

U A | y<sup>j</sup> | too tiny to fit accurately in extended format causes the U exception if U is masked. In this case, mqerY18 returns a gradual underflow denormal, if possible; otherwise, it returns ±0. A | y<sup>j</sup> | less than the smallest positive normalized number causes the U exception if U is unmasked. In this case, control passes to the exception handler with the input y in ST(1), j in ST, and the exception opcode set.

### **Exception Opcode**

27CH

### Example

mqerYI8 raises a real number y on the 80387 stack to the power of a 64-bit integer in the EDX (most significant digits) and EAX (least significant digits) registers. Its results fall in the range of finite, representable numbers or its results are  $\pm \infty$ .

```
: in DATA segment
                       ; initialized
INT POWER DO 9
REAL Y DO 3.3
                        : to test values
REAL OUTPUT DQ ?
                         : in CODE32 segment
EXTRN MQERYI8: NEAR
; The following lines raise REAL Y to INT POWER
; and store the result in REAL OUTPUT.
FLD REAL Y
                         : push. ST := REAL Y
MOV EAX, DWORD PTR INT POWER
                         ; EAX := lower 4 bytes(
                                      INT POWER)
MOV EDX, DWORD PTR INT POWER+4
                         : EDX := upper 4 bytes(
                                      INT POWER)
                         : ST := ST**EDX EAX
CALL MQERYI8
FSTP REAL OUTPUT
                         ; Store result, pop ST.
REAL OUTPUT is now about 46411.5.
```

Returns real number raised to a short integer power

## **Function**

ST := ST(top\_4\_bytes\_80386\_stack) = ST(dword ptr SS:ESP)

Result :=  $y^j$  if  $-\infty \le y \le +\infty$  and  $-2,147,483,648 \le j \le +2,147,483,647$   $(-2^{31} \le j \le +(2^{31}-1))$  as follows:

- Result := 1.0 if i = 0
- Result :=  $\pm 0$  if  $y = \pm 0$ , j > 0, and j is even
- Result := y if  $y = \pm 0$ , j > 0, and j is odd
- Result :=  $y^j$  if  $0 < y < +\infty$  and  $-2^{31} \le j < 0$  or  $0 < j \le +(2^{31} 1)$
- Result :=  $|y|^j$  if  $-\infty < y < 0$ ,  $j \ne 0$ , and j is even
- Result :=  $-(|y|^j)$  if  $-\infty < y < 0$  and j is odd

#### Discussion

mqerYIS returns  $y^{j}$ , where y is a real number and j is a 32-bit integer expressed in two's complement and pushed on the 80386 stack. If j = 0, mqerYIS returns +1.0, whatever the value of y.

For finite y > 0, mqerYIS returns:

- $2^{(j*log\_base\_2(y))}$  if |j| > 63
- $y^j$  if  $1 \le j \le 63$  by successively squaring and multiplying y to compute the correct power
- $y^j$  if  $-63 \le j \le -1$  by evaluating one of two possible expressions:
  - 1 / (y<sup>[ j ]</sup>) provided that the denominator would not cause numeric overflow or underflow
  - (1/y) | j | otherwise

For finite y < 0, mqerYIS first computes  $|y|^j$  as for finite y > 0. Then, it determines the sign of the result according to whether j is even (+) or odd (-).

# mqerYIS (continued)

For  $|j| \le 63$ , mqerYIS performs no more than nine multiplications in evaluating  $(y^*y^*(y^*y) ...^*(y^*...^*y))$  with its squaring-and-multiplying algorithm.

mqerYIS accepts input  $y = \pm \infty$  or  $y = \pm 0$  and returns:

- $+\infty$  if  $y = +\infty$  and j > 0
- +0 if  $y = +\infty$  and j < 0
- $+\infty$  if  $y = -\infty$ , j > 0, and j is even
- $-\infty$  if  $y = -\infty$ , j > 0, and j is odd
- +0 if  $y = -\infty$ , j < 0, and j is even
- -0 if  $y = -\infty$ , j < 0, and j is odd
- +0 if  $y = \pm 0$ , j > 0, and j is even
- y if  $y = \pm 0$ , j > 0, and j is odd

## **Exceptions**

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

- I Input y = SNaN causes the I exception. If I is masked, mqerYIS returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerYIS returns its result. If D is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- Z Input  $y = \pm 0$  with j < 0 causes the Z exception. If Z is masked, mqerYIS returns  $\infty$  with the same sign as y. If Z is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.
- O A y<sup>j</sup> whose exponent is too large to be represented in the extended format causes the O exception. If O is masked, mqerYIS returns  $\infty$  with the same sign as the true result. If O is unmasked, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.

U A  $|y^j|$  too tiny to fit accurately in extended format causes the U exception if U is masked. In this case, mqerYIS returns a gradual underflow denormal, if possible; otherwise, it returns  $\pm 0$ . A  $|y^j|$  less than the smallest positive normalized number causes the U exception if U is unmasked. In this case, control passes to the exception handler with the input y in ST(1), the input j in ST, and the exception opcode set.

### **Exception Opcode**

27CH

## Example

mqerYIS raises a real number on the 80387 stack to the power of a 32-bit integer on the 80386 stack. Its results fall in the range of finite, representable numbers or its results are  $\pm \infty$ .

: in DATA segment

```
INTEREST RATE DO 0.015 ; initialized
MONTHS DD 12
                          : to test
START AMT DO 1000.00
                         : values
FINISH AMT DO ?
                          : in CODE32 segment
EXTRN MOERYIS: NEAR
: The following lines compound interest
: on START AMT for a year at 1.5% per month.
FLD1
                          ; push, ST := 1.0
FADD INTEREST RATE
                          : ST := ST+0.015
PUSH MONTHS
                          : 32-bit integer exponent
                          : pushed onto 80386 stack
CALL MOERYIS
                         : ST := ST**(DWORD PTR SS:ESP)
                         : (ST := 1.015**MONTHS)
FMUL START AMT
                          : ST := ST*START AMT
                          : (Scale by $100\overline{0}.)
FSTP FINISH AMT
                          : Store result, pop ST.
```

; FINISH\_AMT is now about \$1195.62. Note that mqerYIS

; pops the exponent from the 80386 stack.

# 3.3 CL387 Complex Functions

The complex functions are a subset of the Common Elementary Function Library. This section contains:

- A summary of the CL387 complex functions
- Some basic information about complex functions and numbers
- Information about how to read each function's reference pages
- A comprehensive reference for each CL387 complex function in alphanumeric order

## 3.3.1 Summary of CL387 Complex Functions

Table 3-3 summarizes the CL387 complex functions.

Table 3-3 Common Elementary Complex Functions

Compute Logarithms and Exponentials:			
Name	Function	Description	
mqerCLGE	Ln(z)	Natural (base e) logarithms	
mqerCEXP	e <sup>z</sup>	Exponential function	
mqerCC2C	w <sup>z</sup>	Raises complex w to complex power z	
mqerCC2R	z <sup>x</sup>	Raises complex z to real power x	
mqerCR2C	x <sup>z</sup>	Raises real x to complex power z	
mqerCCIS	z <sup>j</sup>	Raises complex z to power of 32-bit integer on 80386 stack	
mqerCCl2	z <sup>j</sup>	Raises complex z to power of 16-bit integer in AX	
mqerCCl4	z <sup>j</sup>	Raises complex z to power of 32-bit integer in EAX	
mqerCCl8	z <sup>j</sup>	Raises complex z to power of 64-bit integer in EDX_EAX	

Table 3-3 Common Elementary Complex Functions (continued)

Compi	ute Trigonom Name	etrics and Hyperbo Function	olics: Description
	mgerCSIN	sin(z)	Trigonometric sine
	mgerCCOS	* *	Trigonometric cosine
	mgerCTAN	tan(z)	Trigonometric tangent
	mqerCASN	* *	Trigonometric inverse sine's principal value
	mqerCACS	Arccos(z)	Trigonometric inverse cosine's principal value
	mqerCATN	Arctan(z)	Trigonometric inverse tangent's principal value
	mgerCSNH	sinh(z)	Hyperbolic sine
	mqerCCSH	cosh(z)	Hyperbolic cosine
	mqerCTNH	tanh(z)	Hyperbolic tangent
	mqerCASH	Arcsinh(z)	Hyperbolic inverse sine's principal value
	mqerCACH	Arccosh(z)	Hyperbolic inverse cosine's principal value
	mqerCATH	Arctanh(z)	Hyperbolic inverse tangent's principal value
Conve	rt Complex I	Representations:	
	Name	Function	Description
	mqerCPOL	(op_rad,op_arg)	Converts rectangular to polar representation
	mqerCREC	(z_re,z_im)	Converts polar to rectangular representation

Table 3-3 Common Elementary Complex Functions (continued)

Return	of Other Values:		
	Name	Function	Description
	mqerCMUL	W * Z	Complex multiplication
	mqerCDIV	w/z	Complex division
	mgerCABS	Z	Complex absolute value (real magnitude)
	mgerCSQR	sqrt(z)	Complex square root
	mqerCPRJ	$z := (\infty, \pm 0)$	Maps square at infinity to (infinity, zero with z_im's sign)

## 3.3.2 Complex Functions and Complex Numbers

Complex functions extend the domain and range of the common realvalued functions to the entire complex number plane.

The field of complex numbers can be represented as plane with real and imaginary axes. Figure 3-2 shows the complex plane with a complex number z in rectangular representation.



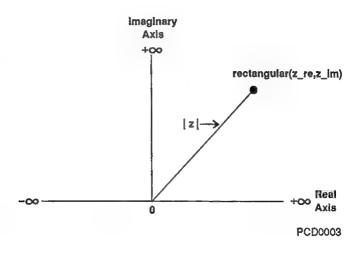


Figure 3-2 Rectangular Graph of a Complex Number

The algebraic representation of a complex number z is x + iy. The imaginary component (i) has the following property:  $i^2 = -1$ .

This chapter uses the rectangular representation  $z = (z_re, z_im)$  where the real and imaginary components of z are an ordered pair of real numbers. Most CL387 complex functions require at least two input arguments in rectangular representation.

This chapter also uses the polar—or circular—representation  $z = (op\_rad, op\_arg)$ , where the radial and angular components of z are an ordered pair of real numbers. Op\\_rad is the magnitude of z, shown graphically as the line segment from the origin to z. Op\_rad is always a non-negative number, defined as the square root of  $(z\_re^2 + z\_im^2)$ . Op\_arg is the principal value of the angle from the positive real axis to the radial line segment. Op\_arg is always an angle expressed in radians such that  $-\pi \le op\_arg \le +\pi$ .

Figure 3-3 shows the complex plane with a complex number z in polar representation.

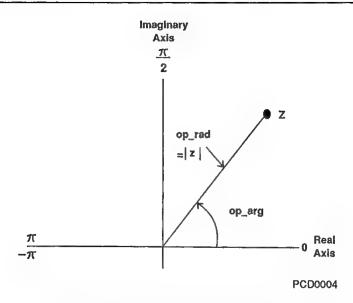


Figure 3-3 Polar Graph of a Complex Number

# 3.3.3 How to Read the Complex Function Reference Pages

The reference pages for each complex function have five sections:

- Function summarizes the result and input arguments for each function in two different ways. The first line uses 80387 stack position notation. Subsequent lines use mathematical notation.
  - For example,  $(ST, ST(1)) := ST(2)^{(ST, ST(1))}$  indicates that a real input in ST(2) is raised to the power of the complex input components in ST and ST(1) and that the components of the complex result are left in ST and ST(1).
  - The second line uses (Re\_Result, Im\_Result), x, and (z\_re, z\_im) to indicate the same thing mathematically, as (Re\_Result, Im\_Result) :=  $x^{(z_re, z_im)} = x^z$ . Subsequent lines sometimes summarize the range for a component of the result.
- 2. Discussion explains the function in more detail. This section uses the mathematical notation of the Function section. Ranges expressed as a..b are inclusive: a and b are part of the range.

- 3. Exceptions first summarizes which exceptions the function can report. Then, it explains what causes each exception and how the function handles the masked and unmasked cases.
- 4. Exception Opcode has the hexadecimal value the function stores in the Opcode field of the 80387 state if the function generates a trap (see Table 3-1).
- 5. Example has a commented ASM386 example that shows how to declare the function for linkage with CL387N.LIB (see Section 3.1.1), how to set up the function's arguments (push order, register load, or both), how to call the function, and how to store the results.

## 3.3.4 CL387 Complex Function Reference

The remaining pages in this section are a comprehensive reference for each Common Elementary Library complex function. The functions are in alphanumeric order.

## mgerCABS

Returns absolute value of complex number

#### **Function**

#### Discussion

mqerCABS returns the absolute value (magnitude) of the complex number z whose components are (z\_re, z\_im). The absolute value is a real number defined mathematically by the equation:

$$|z| =$$
 $|z| =$ 
 $|z| =$ 

The magnitude of a complex number represents the distance in the complex plane between z and the origin as shown in Figure 3-4.

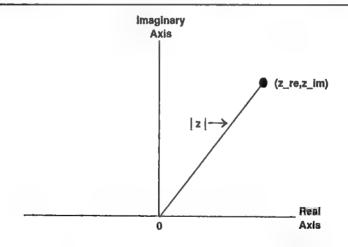


Figure 3-4 Absolute Value of a Complex Number

### **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, Underflow, and Precision:

- I Input SNaNs cause the I exception. If I is masked, mqerCABS returns the QNaN indefinite. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCABS returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O mqerCABS defends against spurious overflow when computing (z\_re² + z\_im²). However, a result whose exponent is too large to fit in extended format causes the O exception. If O is masked, mqerCABS returns +\infty. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- U Computing (z\_re² + z\_im²) where | z\_re | and | z\_im | are both very close to zero can produce an intermediate result whose square root is so tiny that it cannot be represented accurately in extended format. If U is masked, this causes the U exception and mqerCABS returns a gradual underflow denormal, if possible; otherwise, it returns +0. If U is unmasked, a result whose absolute value is too tiny to be represented as a normalized number causes the U exception. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- P An inexact (P exception) might be signaled due to the rounding of intermediate results even though the final result is exact.

  However, if no P exception occurs, the result is exact.

## **Exception Opcode**

283H

# mqerCABS (continued) Example

mqerCABS returns the absolute value of a complex number z with components (z re, z im); its result is an extended format real.

```
: in DATA segment
Z RE DT 3.0
                       ; initialized
Z_IM DT 4.0
                        ; to test values
ABS VAL Z DT ?
                         : in CODE32 segment
EXTRN MQERCABS: NEAR
; The following lines compute the real absolute value
; (magnitude) of the complex number (Z RE, Z IM)
; and store the result in ABS VAL Z.
FLD Z IM
                       : push, ST := Z IM
                    ; push, ST := Z_RE
: ST := | (ST,ST(1)) |
FLD Z RE
CALL MQERCABS
FSTP ABS VAL Z
                      : Store result, pop stack.
; ABS_VAL Z = 5.0
```

## **Function**

(ST, ST(1)) := Arccosh(ST, ST(1))  
(Arccosh\_Re, Arccosh\_Im) := Arccosh(z\_re, z\_im)  
= Arccosh(z) with 
$$-\pi \le Arccosh_Im \le +\pi$$

#### Discussion

mqerCACH returns the principal value of the complex hyperbolic arc cosine for the complex number z with real and imaginary components (z\_re, z\_im). The imaginary component of its result falls in the range  $-\pi..+\pi$ .

The mathematical definition for Arccosh(z) (or cosh<sup>-1</sup> z) is the complex integral:

$$Arccosh(z) = \begin{cases} z & dt \\ \hline t^2 - 1 & \end{cases}$$

The path of integration must not cross the real axis, except possibly in the interval  $+1.0 < z_re < +\infty$ .

Figure 3-5 shows the domain of Arccosh(z) with points B' = (-1,-0),  $Z' = (\pm 0,-0)$ ,  $A = (+1,\pm 0)$ ,  $Z = (\pm 0,+0)$ , and B = (-1,+0) on the branch cuts. Figure 3-6 shows the image of the domain under Arccosh(z).

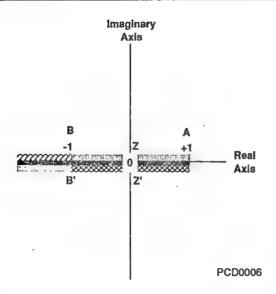


Figure 3-5 Domain of Arccosh(z) with Branch Cuts

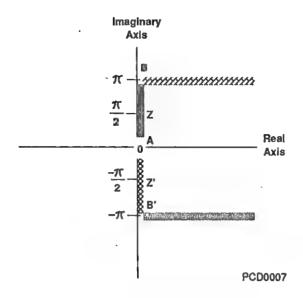


Figure 3-6 Range of Arccosh(z)

## **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCACH returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCACH returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCACH returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCACH returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.

#### **Exception Opcode**

3A2H

# mqerCACH (continued) Example

mqerCACH returns Arccosh(z\_re, z\_im) for the complex number z; the imaginary component of its result falls in the range  $-\pi..+\pi$ .

```
: in DATA segment
                        ; initialized
Z RE DT 1.0
Z IM DT 3.0
                        : to test values
ARCCOSH RE DT ?
ARCCOSH IM DT ?
                         ; in CODE32 segment
EXTRN MQERCACH: NEAR
: The following lines compute Arccosh(Z RE.Z IM) and
; store the real and imaginary components of the
; result in ARCCOSH RE and ARCCOSH IM.
                         ; push, ST := Z IM
FLD Z IM
                         : push, ST := Z RE
FLD Z RE
CALL MQERCACH
                        : (ST,ST(1)) :=
                                      Arccosh(ST.ST(1))
                        ; Store real component, pop ST
FSTP ARCCOSH RE
                         : Store imaginary component.
FSTP ARCCOSH IM
                         ; pop ST.
: ARCCOSH RE is about 1.8642.
; ARCCOSH IM is about 1.2632.
```

#### **Function**

(ST, ST(1)) := Arccos(ST, ST(1))  
(Arccos\_Re, Arccos\_Im) := Arccos(z\_re, z\_im)  
= Arccos(z) with 
$$0 \le Arccos_Re \le +\pi$$

#### Discussion

mqerCACS returns the principal value of the complex arc cosine for the complex number z with real and imaginary components (z\_re, z\_im). The real component of its result falls in the range  $0..+\pi$ .

The mathematical definition for Arccos(z) (or cos<sup>-1</sup> z) is the complex integral:

$$Arccos(z) = \int_{z}^{1} \frac{dt}{1 - t^2}$$

The path of integration must not cross the real axis, except possibly in the interval  $-1.0 < z_re < +1.0$ .

Figure 3-7 shows the domain of Arccos(z) with points C = (x,+0),  $B = (-1,\pm0)$ , C' = (x,-0), D = (x,+0),  $A = (+1,\pm0)$ , and D' = (x,-0) on the branch cuts. Figure 3-8 shows the image of the domain under Arccos(z).

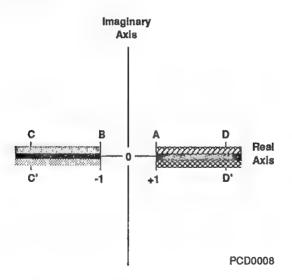


Figure 3-7 Domain of Arccos(z) with Branch Cuts

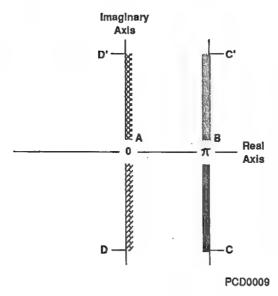


Figure 3-8 Range of Arccos(z)

## **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCACS returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCACS returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCACS returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCACS returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

### **Exception Opcode**

3A1H

# mqerCACS (continued) Example

mqerCACS returns Arccos(z\_re, z\_im) for the complex number z; the real component of its result falls in the range  $0..+\pi$ .

```
; in DATA segment
                       ; initialized ; to test values
Z RE DT 1.0
Z_IM DT 3.0
ARCCOS RE DT ?
ARCCOS IM DT ?
                         ; in CODE32 segment
EXTRN MQERCACS: NEAR
; The following lines compute Arccos(Z RE, Z IM) and
: store the real and imaginary components of the
; result in ARCCOS RE and ARCCOS IM.
FLD Z IM
                         ; push, ST := Z IM
FLD Z RE
                        ; push, ST := Z RE
CALL mqerCACS
                       ; (ST,ST(1)) :=
                                     Arccos(ST,ST(1))
                      ; Store real component, pop ST
FSTP ARCCOS RE
                        ; Store imaginary component,
FSTP ARCCOS IM
                        : pop ST.
: ARCCOS RE is about 1.2632.
; ARCCOS IM is about -1.8642.
```

hyperbolic Arc sine

## **Function**

(ST, ST(1)) := Arcsinh(ST, ST(1))  
(Arcsinh\_Re, Arcsinh\_Im) := Arcsinh(z\_re, z\_im)  
= Arcsinh(z) with 
$$-\pi/2 \le Arcsinh$$
 Im  $\le +\pi/2$ 

#### Discussion

mqerCASH returns the principal value of the complex hyperbolic arc sine for the complex number z with real and imaginary components (z\_re, z\_im). The imaginary component of its result falls in the range  $-\pi/2..+\pi/2$ .

The mathematical definition for Arcsinh(z) (or sinh<sup>-1</sup> z) is the complex integral:

The path of integration must not cross the imaginary axis, except possibly in the interval  $-1.0 < z_im < +1.0$ .

Figure 3-9 shows the domain of Arcsinh(z) with the points C = (-0,x), B = (-0,-1), A = (+0,+1), and D = (+0,x) on the branch cuts. Figure 3-10 shows the image of the domain under Arcsinh(z).

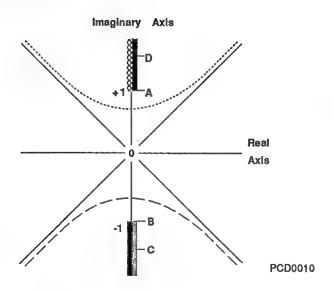


Figure 3-9 Domain of Arcsinh(z) with Branch Cuts

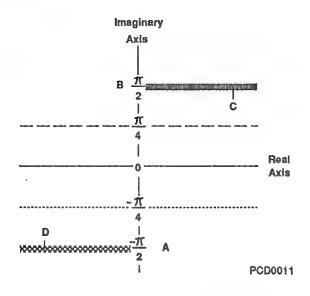


Figure 3-10 Range of Arcsinh(z)

## **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCASH returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), the input z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCASH returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCASH returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCASH returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

## **Exception Opcode**

3A0H

# mqerCASH (continued) Example

mqerCASH returns Arcsinh(z\_re, z\_im) for the complex number z; the imaginary component of its result falls in the range  $-\pi/2..+\pi/2$ .

```
; in DATA segment
                        ; initialized
Z RE DT 1.0
Z IM DT 3.0
                        : to test values
ARCSINH RE DT ?
ARCSINH IM DT ?
                        : in CODE32 segment
EXTRN mgerCASH: NEAR
; The following lines compute Arcsinh(Z RE, Z IM) and
: store the real and imaginary components of the
; result in ARCSINH RE and ARCSINH IM.
                         ; push, ST := Z IM
FLD Z IM
                         ; push, ST := Z RE
FLD Z RE
                         ; (ST,ST(1)) :=
CALL mgerCASH
                                      Arcsinh(ST,ST(1))
                        ; Store real component, pop ST
FSTP ARCSINH RE
FSTP ARCSINH IM
                         ; Store imaginary component,
                         : pop ST.
; ARCSINH RE is about 1.8242.
: ARCSINH IM is about 1.2331.
```

#### **Function**

(ST, ST(1)) := Arcsin(ST, ST(1))  
(Arcsin\_Re, Arcsin\_Im) := Arcsin(z\_re, z\_im)  
= Arcsin(z) with 
$$-\pi/2 \le Arcsin_Re \le +\pi/2$$

#### Discussion

mqerCASN returns the principal value of the complex arc sine for the complex number z with real and imaginary components (z\_re, z\_im). The real component of its result falls in the range  $-\pi/2..+\pi/2$ .

The mathematical definition for Arcsin(z) (or sin<sup>-1</sup> z) is the complex integral:

The path of integration must not cross the real axis, except possibly in the interval -1.0 < z re < +1.0

Figure 3-11 shows the domain of Arcsin(z) with the points C = (x,+0), B = (-1,+0), A = (+1,-0), and D = (x,-0) on the branch cuts. Figure 3-12 shows the image of the domain under Arcsin(z).

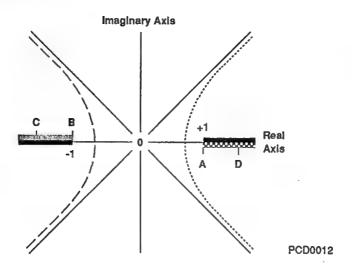
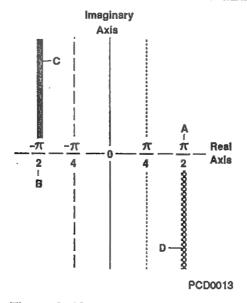


Figure 3-11 Domain of Arcsin(z) with Branch Cuts



#### **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCASN returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), the input z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCASN returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCASN returns ± to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCASN returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# **Exception Opcode**

39FH

# mqerCASN (continued) Example

mqerCASN returns Arcsin(z\_re, z\_im) for the complex number z; the real component of its result falls in the range  $-\pi/2..+\pi/2$ .

```
: in DATA segment
        :
Z RE DT 1.0
                       : initialized
Z_IM DT 3.0
                         : to test values
ARCSIN RE DT ?
ARCSIN IM DT ?
                         : in CODE32 segment
EXTRN MQERCASN: NEAR
: The following lines compute Arcsin(Z RE.Z IM) and
: store the real and imaginary components of the
: result in ARCSIN RE and ARCSIN IM.
                         : push, ST := Z IM
FLD Z IM
FLD Z RE
                         : push. ST := Z RE
                        : (ST,ST(1)) :=
CALL MQERCASN
                                     Arcsin(ST,ST(1))
                        Store real component, pop ST
FSTP ARCSIN RE
                         Store imaginary component.
FSTP ARCSIN IM
                         : pop ST.
: ARCSIN RE is about 0.3076.
: ARCSIN IM is about 1.8642.
```

#### **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCASN returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), the input z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCASN returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCASN returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCASN returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

## **Exception Opcode**

39FH

# mqerCASN (continued) Example

mqerCASN returns Arcsin(z\_re, z\_im) for the complex number z; the real component of its result falls in the range  $-\pi/2..+\pi/2$ .

```
: in DATA segment
                        : initialized
Z RE DT 1.0
                         : to test values
Z IM DT 3.0
ARCSIN RE DT ?
ARCSIN IM DT ?
                         ; in CODE32 segment
EXTRN MQERCASN: NEAR
: The following lines compute Arcsin(Z RE, Z IM) and
: store the real and imaginary components of the
; result in ARCSIN RE and ARCSIN IM.
                         ; push, ST := Z IM
FLD Z IM
FLD Z RE
                         : push, ST := Z RE
CALL MQERCASN
                        ; (ST,ST(1)) :=
                                      Arcsin(ST,ST(1))
                        ; Store real component, pop ST
FSTP ARCSIN RE
FSTP ARCSIN IM
                         : Store imaginary component,
                         ; pop ST.
: ARCSIN RE is about 0.3076.
: ARCSIN IM is about 1.8642.
```

#### **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCATH returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCATH returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCATH returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCATH returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.

#### **Exception Opcode**

3A4H

# mqerCATH (continued) Example

mqerCATH returns Arctanh(z\_re, z\_im) for the complex number z; the imaginary component of its result falls in the range  $-\pi/2..+\pi/2$ .

```
: in DATA segment
Z_RE DT 1.0
Z_IM DT 3.0
                       ; initialized
                        : to test values
ARCTANH RE DT ?
ARCTANH IM DT ?
                        : in CODE32 segment
EXTRN MOERCATH: NEAR
: The following lines compute Arctanh(Z RE.Z IM) and
; store the real and imaginary components of the
; result in ARCTANH RE and ARCTANH IM.
FLD Z IM
                         ; push, ST := Z IM
                         : push, ST := Z RE
FLD Z RE
                        ; (ST,ST(1)) :=
CALL MOERCATH
                                      Arctanh(ST,ST(1))
FSTP ARCTANH RE
                        ; Store real component, pop ST
FSTP ARCTANH IM
                         ; Store imaginary component,
                         ; pop ST.
: ARCTANH RE is about 0.0919.
: ARCTANH IM is about 1.2768.
```

#### **Function**

(ST, ST(1)) := Arctan(ST, ST(1))  
(Arctan\_Re, Arctan\_Im) := Arctan(z\_re, z\_im)  
= Arctan(z) with 
$$-\pi/2 \le Arctan_Re \le +\pi/2$$

#### Discussion

mqerCATN returns the principal value of the complex arc tangent for the complex number z with real and imaginary components (z\_re, z im). The real component of its result falls in the range  $-\pi/2..+\pi/2$ .

mqerCATN accepts infinite arguments and returns ( $\pi/2$  with z\_re's sign, 0 with z\_im's sign).

The mathematical definition for Arctan(z) (or tan<sup>-1</sup> z) is the complex integral:

$$Arctan(z) = \begin{cases} z & dt \\ \frac{1+t^2}{1} \end{cases}$$

The path of integration must not cross the imaginary axis, except possibly in the interval -1.0 < z im < +1.0.

Figure 3-15 shows the domain of Arctan(z) with the points A' = (+0,+1), C' =  $(+1,\pm0)$ , B' = (+0,-1), B = (-0,-1), C =  $(-1,\pm0)$ , and A = (-0,+1) on the branch cuts. Figure 3-16 shows the image of the domain under Arctan(z).

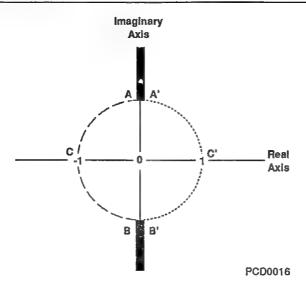
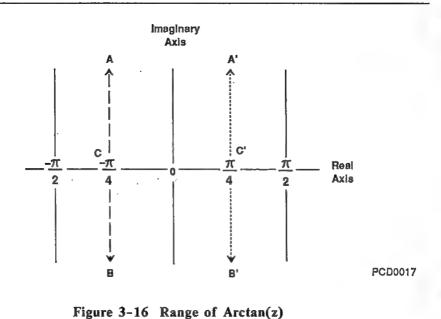


Figure 3-15 Domain of Arctan(z) with Branch Cuts



## **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCATN returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCATN returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCATN returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCATN returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.

#### **Exception Opcode**

3A3H

# mqerCATN (continued) Example

mqerCATN returns Arctan(z\_re, z\_im) for the complex number z; the real component of its result falls in the range  $-\pi/2..+\pi/2$ .

```
: in DATA segment
                        : initialized
7 RF DT 1.0
Z IM DT 3.0
                         : to test values
ARCTAN RE DT ?
ARCTAN IM DT ?
                         : in CODE32 segment
EXTRN MOERCATN: NEAR
: The following lines compute Arctan(Z RE.Z IM) and
: store the real and imaginary components of the
: result in ARCTAN RE and ARCTAN IM.
                         : push, ST := Z IM
FLD Z IM
FLD Z RE
                         ; push, ST := Z RE
                         : (ST.ST(1)) :=
CALL MOERCATN
                                      Arctan(ST,ST(1))
                         : Store real component, pop ST
FSTP ARCTAN RE
                         : Store imaginary component,
FSTP ARCTAN IM
                          : pop ST.
: ARCTAN RE is about 1.4615.
: ARCTAN IM is about 0.3059.
```

Returns complex number raised to complex power

# **Function**

$$(ST, ST(1)) := (ST(2), ST(3))^{(ST, ST(1))}$$
  
 $(Re\_Result, Im\_Result) := (w\_re, w\_im)^{(z\_re, z\_îm)} = w^z$ 

#### Discussion

mqerCC2C returns the complex result of raising a complex number w to a complex power z. mqerCC2C returns:

- e z\*tn(w) if z\_im ≠ ±0, according to the calculations of mqerCMUL, mqerCLGE, and mqerCEXP
- w<sup>z\_re</sup> if z\_im = ±0, according to the calculation of mqerCC2R

# Exceptions

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCC2C returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCC2C returns its result. If D is unmasked, control passes to the exception handler with w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- Z Input z\_re < 0 with w\_re, w\_im, and z\_im = ±0 causes the Z exception. If Z is masked, mqerCC2C returns ±∞ to affected components of the result. If Z is unmasked, control passes to the exception handler with w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

## mgerCC2C (continued)

- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCC2C returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCC2C returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

## **Exception Opcode**

58EH

#### Example

mqerCC2C returns the complex result of raising one complex number to the power of another.

```
; in DATA segment
W RE DT 1.0
                         : initialized
WIM DT 0.5
                         ; to
Z RE DT 1.0
                         : test
Z IM DT 1.0
                         : values
RE RESULT DT ?
IM RESULT DT ?
                         ; in CODE32 segment
EXTRN MOERCC2C: NEAR
; The following lines compute
; (W RE, W IM)**(Z RE, Z IM) and store the real
; and imaginary components of the result
; in RE RESULT and IM RESULT.
FLD W IM
                         ; push, ST := W IM
FLD W RE
                         ; push, ST := W RE
                         ; push, ST := Z IM
FLD Z IM
FLD Z RE
                         : push, ST := Z RE
                         : (ST,ST(1)) := (ST(2),ST(3))
CALL MOERCC2C
                                      **(ST, ST(1))
FSTP RE RESULT
                         ; Store real component, pop ST
FSTP IM RESULT
                         : Store imaginary component,
                         ; pop ST.
: RE RESULT is about 0.5901.
; IM RESULT is about 0.3826.
```

# mgerCC2R

Returns complex number raised to real power

#### **Function**

$$(ST, ST(1)) := (ST(1), ST(2))^{ST}$$
  
(Re\_Result, Im\_Result) := (z\_re, z\_im)^x = z^x

#### Discussion

mqerCC2R returns the complex result of raising a complex number z to the power of a real number x. If  $x = \pm 0$ , mqerCC2R returns (+1,  $x^*(1 \text{ with } z_im's \text{ sign})$ ), whatever the value of z.

For non-integer x or  $x = \pm \infty$ , mqerCC2R returns:

- e (x\*ln(| z |), x\*principal\_value\_angle(z\_re, z\_im)) if z im ≠ ±0
- $(z_re^x, z_im^*(1 \text{ with } x^s \text{ sign})) \text{ if } z_re \ge 0 \text{ and } z_im = \pm 0$
- $e^{(x*ln(|z|), (x \mod 2)*principal_value_angle(z_re, z_im))}$ if z re not  $\geq 0$  and z im =  $\pm 0$

The function principal\_value\_angle returns the angle between the positive real axis and the line segment connecting the origin to the point z (see mqerCPOL).

For nonzero, integer x, mqerCC2R returns:

- $e^{x^* \ln(z)}$  if |x| > 8 and  $z_{im} \neq 0$
- $(z_re^x, x*z_im*((1 \text{ with } z_re^s \text{ sign})*(1 \text{ with } z_re^x \text{ sign})))$ if |x| > 8 and  $z_im = \pm 0$
- $z^x$  if  $1 \le x \le 8$  by successively squaring and multiplying z to compute the correct power
- $z^x$  if  $-8 \le x \le -1$  by evaluating one of two possible expressions:
  - 1 / (z | x |) provided that the denominator would not cause numeric overflow or underflow
  - $(1/z)^{|x|}$  otherwise

## **Exceptions**

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow.

- I Input SNaNs cause the I exception. If I is masked, mqerCC2R returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(2), z\_re still in ST(1), x still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCC2R returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(2), z\_re still in ST(1), x still in ST, and the exception opcode set.
- Z Input x < 0 with z\_re and z\_im = ±0 causes the Z exception. If Z is masked, mqerCC2R returns ±∞ to affected components of the result. If Z is unmasked, control passes to the exception handler with z\_im still in ST(2), z\_re still in ST(1), x still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCC2R returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(2), z\_re still in ST(1), x still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCC2R returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(2), z\_re still in ST(1), x still in ST, and the exception opcode set.

## **Exception Opcode**

592H

# mqerCC2R (continued) Example

mqerCC2R returns the complex result of raising a complex number to a real power.

```
; in DATA segment
Z RE DT 1.0
                         ; initialized
Z IM DT 0.5
                        : to
X DT 2.0
                         : test values
RE RESULT DT ?
IM RESULT DT ?
                         : in CODE32 segment
    : :
EXTRN MQERCC2R: NEAR
; The following lines compute (Z RE, Z IM)**X and
; store the real an imaginary components
; of the result in RE RESULT and IM RESULT.
FLD Z IM
                         ; push, ST := Z IM
                         ; push, ST := Z RE
FLD Z RE
                         ; push, ST := X
FLD X
                         : (ST,ST(1)) := (ST(2),ST(1))
CALL MQERCC2R
                                     **ST
FSTP RE RESULT
                        ; Store real component, pop ST
FSTP IM RESULT
                         : Store imaginary component.
                         ; pop ST.
: RE RESULT is about 0.75.
: IM RESULT is about 1.0.
```

Returns complex number raised to word integer power

## **Function**

$$(ST, ST(1)) := (ST, ST(1))^{AX}$$
  
 $(Re\_Result, Im\_Result) := (z\_re, z\_im)^j$  for -32768 \le j \le +32767

#### Discussion

mqerCCI2 returns the complex result of raising the complex number z to the power of a 16-bit integer j, expressed in two's complement.

If j = 0, mqerCCI2 returns (+1, 0), whatever the value of z. Otherwise, mqerCCI2 returns:

- $e^{j*ln(z)}$  if |j| > 8 and  $z_{im} \neq 0$
- $(z_re^j, j*z_im*((1 with z_re^i s sign)*(1 with z_re^i s sign)))$ if |j| > 8 and  $z_im = \pm 0$
- $z^j$  if  $1 \le j \le 8$  by successively squaring and multiplying z to compute the correct power
- $z^{j}$  if  $-8 \le j \le -1$  by evaluating one of two possible expressions:
  - 1 / (z<sup>| j |</sup>) provided that the denominator would not cause numeric overflow or underflow
  - $(1/z)^{|j|}$  otherwise

#### **Exceptions**

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

I Input SNaNs cause the I exception. If I is masked, mqerCCI2 returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.

## mgerCCI2 (continued)

- D Input denormals cause the D exception. If D is masked, mqerCCI2 returns its result. If D is unmasked, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.
- Z Input j < 0 with z\_re and z\_im = ±0 causes the Z exception. If Z is masked, mqerCCI2 returns ±∞ to affected components of the result. If Z is unmasked, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCCl2 returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCCI2 returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im in ST(2), z re in ST(1), j in ST, and the exception opcode set.

# **Exception Opcode**

595H

## Example

mqerCCI2 returns the complex result of raising a complex number on the 80387 stack to the power of an integer in the range -32768..32767. The input exponent is in the AX register.

```
: in DATA segment
Z RE DT 1.0
                         : initialized
Z IM DT 0.5
                         : to
J DW 02FFH
                         : test values
RE RESULT DT
IM RESULT DT ?
                         ; in CODE32 segment
EXTRN MOERCCI2: NEAR
; The following lines compute (Z RE, Z IM)**J and
; store the real and imaginary components
; of the result in RE RESULT and IM RESULT.
FLD Z IM
                          : push, ST := Z IM
FLD Z RE
                         ; push, ST := Z RE
MOV AX. J
                         : AX := J
CALL MOERCCI2
                          : (ST,ST(1)) := (ST,ST(1))**AX
FSTP RE RESULT
                         ; Store real component, pop ST
FSTP
     IM RESULT
                         ; Store imaginary component,
                          ; pop ST.
; RE RESULT is about -1.1919E37.
; IM RESULT is about -8.4687E36.
```

# mgerCCI4

Returns complex number raised to short integer power

#### **Function**

(ST, ST(1)) := (ST, ST(1))<sup>EAX</sup> (Re\_Result, Im\_Result) := 
$$(z_re, z_im)^j$$
 for  $-2^{31} \le j \le +(2^{31} - 1)$ 

#### Discussion

mqerCCI4 returns the complex result of raising the complex number z to the power of a 32-bit integer j, expressed in two's complement.

If j = 0, mqerCCI4 returns (+1, 0), whatever the value of z. Otherwise, mqerCCI4 returns:

- $e^{j*in(z)}$  if |j| > 8 and z im  $\neq 0$
- (z\_re<sup>j</sup>, j\*z\_im\*((1 with z\_re's sign)\*(1 with z\_re<sup>j</sup>'s sign)))
   if | j | > 8 and z im = ±0
- $z^j$  if  $1 \le j \le 8$  by successively squaring and multiplying z to compute the correct power
- $z^{j}$  if  $-8 \le j \le -1$  by evaluating one of two possible expressions:
  - 1 / (z | j |) provided that the denominator would not cause numeric overflow or underflow
  - $(1/z)^{|j|}$  otherwise

## **Exceptions**

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

I Input SNaNs cause the I exception. If I is masked, mqerCC14 returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.

# mqerCCI4 (continued)

- D Input denormals cause the D exception. If D is masked, mqerCCI4 returns its result. If D is unmasked, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.
- Z Input j < 0 with z\_re and z\_im = ±0 causes the Z exception. If Z is masked, mqerCCI4 returns ±∞ to affected components of the result. If Z is unmasked, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCCI4 returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im in ST(2), z re in ST(1), j in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCCI4 returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.

#### **Exception Opcode**

595H

# mqerCCI4 (continued) Example

mqerCCI4 returns the complex result of raising a complex number on the 80387 stack to the power of an integer in the range  $-2^{31}$ ..+ $(2^{31} - 1)$ . The input exponent is in the EAX register.

```
: in DATA segment
                         : initialized
Z RE DT 1.0
Z IM DT 0.5
                        : to
J DD 00000210H
                        : test values
RE RESULT DT
IM RESULT DT
                         : in CODE32 segment
EXTRN MOERCCI4: NEAR
; The following lines compute (Z RE, Z IM)**J and
: store the real and imaginary components
; of the result in RE RESULT and IM RESULT.
FLD Z IM
                         ; push, ST := Z IM
FLD Z RE
                         ; push, ST := Z RE
                         ; EAX := J
MOV EAX. J
CALL MQERCC14
                         : (ST,ST(1)) := (ST,ST(1))
                                      **EAX
                        ; Store real component, pop ST
FSTP RE RESULT
                         : Store imaginary component.
FSTP IM RESULT
                         : pop ST.
: RE RESULT is about 3.7307E25.
: IM RESULT is about -9.0621E24.
```

Returns complex number raised to long integer power

#### **Function**

(ST, ST(1)) := (ST, ST(1))<sup>EDX\_EAX</sup>  
(Re\_Result, Im\_Result) := 
$$(z_re, z_im)^j$$
 for  $-2^{63} \le j \le +(2^{63} - 1)$ 

#### Discussion

mqerCCI8 returns the complex result of raising the complex number z to power of a 64-bit integer j, expressed in two's complement

If j = 0, mqerCCI8 returns (+1, 0), whatever the value of z. Otherwise, mqerCCI8 returns:

- $e^{j*ln(z)}$  if |j| > 8 and z im  $\neq 0$
- $(z_re^j, j*z_im*((1 \text{ with } z_re^i\text{s sign})*(1 \text{ with } z_re^j\text{s sign})))$ if |j| > 8 and  $z_im = \pm 0$
- $z^j$  if  $1 \le j \le 8$  by successively squaring and multiplying z to compute the correct power
- $z^{j}$  if  $-8 \le j \le -1$  by evaluating one of two possible expressions:
  - 1 / (z | j |) provided that the denominator would not cause numeric overflow or underflow
  - $(1/z)^{|j|}$  otherwise

#### **Exceptions**

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

I Input SNaNs cause the I exception. If I is masked, mqerCCI8 returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.

#### mgerCCI8 (continued)

- D Input denormals cause the D exception. If D is masked, mqerCCI8 returns its result. If D is unmasked, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.
- Z Input j < 0 with z\_re and z\_im = ±0 causes the Z exception. If Z is masked, mqerCC18 returns ±∞ to affected components of the result. If Z is unmasked, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.</p>
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCCI8 returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCCI8 returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set

## **Exception Opcode**

595H

#### Example

mqerCCI8 returns the complex result of raising a complex number on the 80387 stack to the power of an integer in the range  $-2^{63}$ ..+ $(2^{63}$  – 1). The input exponent is in the EDX (most significant digits) and EAX registers.

```
; in DATA segment
                         ; initialized
Z RE DT 1.0
                         ; to test
Z_IM DT 0.5
J DO 000000000000122H : values
RE RESULT DT ?
IM RESULT DT ?
                         : in CODE32 segment
EXTRN MOERCCI8: NEAR
; The following lines compute (Z RE, Z IM)**EDX EAX
; and store the real and imaginary components
; of the result in RE RESULT and IM RESULT.
FLD Z IM
                         ; push, ST := Z IM
                         ; push, ST := Z RE
FLD Z RE
MOV EAX, J
                         : EAX := lower 4 bytes(J)
                         ; EDX := upper 4 bytes(J)
MOV EDX, J+4
                         : (ST,ST(1)) := (ST,ST(1))
CALL MOERCCI8
                                      **EDX EAX
FSTP
     RE RESULT
                         : Store real component, pop ST
FSTP IM RESULT
                         ; Store imaginary component,
                         : pop ST.
: RE RESULT is about -9.1026E13.
: IM RESULT is about 6.6463E13.
```

# maerCCIS

Returns complex number raised to short integer power

#### **Function**

$$(ST, ST(1)) := (ST, ST(1))^{(top_4_bytes_80386_stack)} = (ST, ST(1))^{(dword ptr SS:ESP)}$$
  
 $(Re_Result, Im_Result) := (z_re, z_im)^j \text{ for } -2^{31} \le j \le +(2^{31} - 1)$ 

## Discussion

mqerCCIS returns the complex result of raising the complex number z on the 80387 stack to the power of a 32-bit integer j, expressed in two's complement and pushed on the 80386 stack.

If j = 0, mqerCCIS returns (+1, 0), whatever the value of z. Otherwise, mqerCCIS returns:

- $e^{j*ln(z)}$  if |j| > 8 and z im  $\neq 0$
- $(z_re^j, j*z_im*((1 \text{ with } z_re^s \text{ sign})*(1 \text{ with } z_re^j*s \text{ sign})))$ if |j| > 8 and  $z_im = \pm 0$
- $z^j$  if  $1 \le j \le 8$  by successively squaring and multiplying z to compute the correct power
- $z^j$  if  $-8 \le j \le -1$  by evaluating one of two possible expressions:
  - 1 / (z | j |) provided that the denominator would not cause numeric overflow or underflow
  - $(1/z)^{|j|}$  otherwise

## **Exceptions**

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

I Input SNaNs cause the I exception. If I is masked, mqerCCIS returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.

## mgerCCIS (continued)

- D Input denormals cause the D exception. If D is masked, mqerCCIS returns its result. If D is unmasked, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.
- Z Input j < 0 with z\_re and z\_im = ±0 causes the Z exception. If Z is masked, mqerCCIS returns ±∞ to affected components of the result. If Z is unmasked, control passes to the exception handler with z\_im in ST(2), z\_re in ST(1), j in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCCIS returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im in ST(2), z re in ST(1), j in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCCIS returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im in ST(2), z re in ST(1), j in ST, and the exception opcode set.

#### **Exception Opcode**

595H

# mqerCCIS (continued) Example

mqerCCIS returns the complex result of raising a complex number on the 80387 stack to the power of an integer in the range  $-2^{31}$ ..+ $(2^{31} - 1)$ . The input exponent is on the 80386 stack.

```
: in DATA segment
                         ; initialized
Z RE DT 1.0
                         ; to
Z IM DT 0.5
J DD 000002FFH
                         : test values
RE RESULT DT
IM RESULT DT
                          ; in CODE32 segment
EXTRN MQERCCIS: NEAR
; The following lines compute (Z RE, Z IM)**J and
: store the real and imaginary components
: of the result in RE RESULT and IM RESULT.
                         : push, ST := Z IM
FLD Z IM
FLD Z RE
                         ; push, ST := Z RE
                         : push J onto 80386 stack
PUSH J
                         : (ST,ST(1)) := (ST,ST(1))**
CALL MQERCCIS
                                      (DWORD PTR SS:ESP)
                         : Store real component, pop ST
FSTP RE RESULT
                         : Store imaginary component.
FSTP IM RESULT
                          : pop ST.
: RE RESULT is about -1.1919E37.
: IM RESULT is about -8.4687E36.
: MQERCCIS pops J from the 80386 stack.
```

#### **Function**

$$(ST, ST(1)) := cos(ST, ST(1))$$
  
 $(Cos_Re, Cos_Im) := cos(z_re, z_im) if -\infty < z_re < +\infty$ 

#### Discussion

mqerCCOS returns the complex cosine of the complex number z with real and imaginary components (z re, z im).

Cos(z) is defined mathematically in terms of the transcendental constant e (2.71828182845904523536..):

$$\cos(z) = \frac{e^{iz} + e^{-iz}}{2}$$

The real and imaginary components are defined in terms of real functions:

$$cos(z) = cos(z_re, z_im) = (Cos_Re, Cos_Im)$$

where

$$Cos_Re = cos(z_re) * cosh(z_im)$$

$$Cos_{m} = -sin(z_{e}) * sinh(z_{i})$$

#### **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

I Input SNaNs and z\_re = ±∞ cause the I exception. If I is masked, mqerCCOS returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

## mgerCCOS (continued)

- D Input denormals cause the D exception. If D is masked, mqerCCOS returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCCOS returns ± to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCCOS returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.

# **Exception Opcode**

39BH

## mqerCCOS (continued)

## Example

mqerCCOS returns cos(z\_re, z\_im) for the complex number z.

```
: in DATA segment
Z RE DT 1.0
                         : initialized
Z IM DT 3.0
                        : to test values
COS RE DT ?
COS IM DT ?
                         : in CODE32 segment
EXTRN MQERCCOS: NEAR
; The following lines compute cos(Z RE, Z IM) and
; store the real and imaginary components
; of the result in COS RE and COS IM.
                         : push. ST := Z IM
FLD Z IM
FLD Z RE
                         : push, ST := Z RE
CALL MOERCCOS
                         : (ST,ST(1)) := cos(ST,ST(1))
                         ; Store real component, pop ST
FSTP COS RE
FSTP COS IM
                         ; Store imaginary component,
                         ; pop ST.
: COS RE is about 5.4396.
; COS IM is about -8.4298.
```

# mgerCCSH

Returns complex hyperbolic cosine

#### **Function**

(ST, ST(1)) := 
$$cosh(ST, ST(1))$$
  
(Cosh\_Re, Cosh\_Im) :=  $cosh(z_re, z_im)$  if  $-\infty < z_im < +\infty$ 

#### Discussion

mqerCCSH returns the complex hyperbolic cosine of the complex number z with real and imaginary components (z\_re, z\_im).

Cosh(z) is defined mathematically in terms of the transcendental constant e (2.71828182845904523536..):

$$\cosh(z) = \frac{e^z + e^{z}}{2}$$

The real and imaginary components are defined in terms of real functions:

$$cosh(z) = cosh(z_re, z_im) = (Cosh_Re, Cosh_im)$$

where

$$Cosh_Re = cosh(z_re) * cos(z_im)$$

$$Cosh_im = sinh(z_re) * sin(z_im)$$

# **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

I Input SNaNs and z\_im = ±∞ cause the I exception. If I is masked, mqerCCSH returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# mqerCCSH (continued)

- D Input denormals cause the D exception. If D is masked, mqerCCSH returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCCSH returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCCSH returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

## **Exception Opcode**

39CH

# mqerCCSH (continued) Example

mqerCCSH returns cosh(z\_re, z\_im) for the complex number z.

```
; in DATA segment
Z RE DT 1.0
                         : initialized
                         : to test values
Z IM DT 3.0
COSH RE DT ?
COSH IM DT ?
                         : in CODE32 segment
EXTRN MQERCCSH: NEAR
: The following lines compute cosh(Z RE, Z IM) and
: store the real and imaginary components
; of the result in COSH RE and COSH IM.
                         ; push, ST := Z IM
FLD Z IM
                         ; push, ST := Z RE
FLD Z RE
                         ; (ST,ST(1)) := cosh(ST,ST(1))
CALL MOERCCSH
                         ; Store real component, pop ST
FSTP COSH RE
FSTP COSH IM
                         : Store imaginary component,
                         : pop ST.
: COSH RE is about -1.5276.
: COSH IM is about 0.1658.
```

# **Function**

$$(ST, ST(1)) := ((ST(2), ST(3)) / (ST, ST(1)))$$
  
 $(Quo\_Re, Quo\_Im) := ((w\_re, w\_im) / (z\_re, z\_im)) = w/z$ 

#### Discussion

mqerCDIV returns the complex quotient of two complex numbers w and z with real and imaginary components (w\_re, w\_im) and (z\_re, z\_im).

The mathematical definition of the complex quotient is as follows:

where

Quo\_Re = 
$$\frac{(w_re * z_re) + (w_im * z_im)}{z_re^2 + z_im^2}$$

For example, (-1, 0) / (0, 1) = (0, 1). Note that input z should not be  $(\pm 0, \pm 0)$ .

# mgerCDIV (continued)

The input w and/or z can have one or more infinity components. mgerCDIV calculates results as follows:

- If w's components are finite and z has one or more infinite components, mqerCDIV replaces each component of w with 0 of the same sign.
- If z's components are finite and w has one or more infinite components, mqerCDIV leaves z's components unchanged.
- Next, it replaces an argument's infinite component(s) with 1 of the same sign and any finite component of this argument with 0 of the same sign.
- Then, it computes the quotient and replaces each nonzero component of the result with an infinity of the same sign.

# **Exceptions**

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

- I Input SNaNs and  $w = z = (\pm 0, \pm 0)$  cause the I exception. If I is masked, mqerCDIV returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCDIV returns its result. If D is unmasked, control passes to the exception handler with the input w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- Z Input  $w \neq (\pm 0, \pm 0)$  with  $z = (\pm 0, \pm 0)$  causes the Z exception. If Z is masked, mqerCDIV returns  $\pm \infty$  to affected components of the result. If Z is unmasked, control passes to the exception handler with the input w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCDIV returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with the input w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z re still in ST, and the exception opcode set.

# mqerCDIV (continued)

U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCDIV returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with the input w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# **Exception Opcode**

58DH

# mqerCDIV (continued) Example

mqerCDIV returns the quotient of two complex numbers.

```
: in DATA segment
W RE DT 1.0
                         : initialized
W IM DT 2.0
                         ; to
Z RE DT -3.0
                         ; test
Z IM DT 4.5
                         : values
OUO RE DT ?
OUO IM DT ?
                         : in CODE32 segment
EXTRN MQERCDIV: NEAR
; The following lines compute
; (W RE, W IM) / (Z RE, Z IM) and
; store the real and imaginary components
; of the result in QUO RE and QUO IM.
FLD W IM
                         : push. ST := W IM
FLD W RE
                         : push, ST := W RE
FLD Z IM
                         : push, ST := Z IM
FLD Z RE
                         ; push, ST := Z RE
CALL MOERCDIV
                         : (ST,ST(1)) := (ST(2),ST(3))
                                      / (ST,ST(1))
                         ; Store real result, pop ST
FSTP QUO RE
     QUO IM
                         : Store imaginary result.
FSTP
                         : pop ST.
; QUO RE is about 0.2051.
: QUO IM is about -0.3590.
```



$$(ST, ST(1)) := e^{(ST, ST(1))}$$
  
 $(Exp Re, Exp Im) := e^{(z_re, z_{im})} = e^{z}$ 

#### Discussion

mqerCEXP returns the complex value of the transcendental constant e (2.71828182845904523536..) raised to the complex power z with real and imaginary components (z re, z im).

mgerCEXP accepts z re = ±\infty and returns:

- $(+\infty, 0 \text{ with z im's sign})$  for z re =  $+\infty$
- (+0, 0 with z im's sign) for z re =  $-\infty$

The exponential of a complex number is defined mathematically as follows:

$$\exp(z) = e^{z_r^e} * (\cos(z_i^m) + (i * \sin(z_i^m)))$$
  
=  $(\exp_Re, \exp_l^m)$ 

where

$$Exp_Re = e^{z_re} * cos(z_im)$$

$$Exp_im = e^{z_re} * sin(z_im)$$

# **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

I Input SNaNs and z\_im = ±∞ cause the I exception. If I is masked, mqerCEXP returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# mgerCEXP (continued)

- D Input denormals cause the D exception. If D is masked, mqerCEXP returns its result. If D is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCEXP returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCEXP returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# **Exception Opcode**

398H

## Example

mqerCEXP returns e raised to the power of the complex number z.

```
: in DATA segment
Z RE DT 1.0
                         : initialized
Z IM DT 3.0
                         : to test values
EXP RE DT ?
EXP IM DT ?
                          ; in CODE32 segment
EXTRN MOEREXP: NEAR
; The following lines compute the complex exponential
: (e^{**}z) of the complex number (Z RE, Z IM) and
; store the real and imaginary components
; of the result in EXP RE and EXP IM.
                          ; push, ST := Z IM
FLD Z IM
                         ; push, ST := Z RE
FLD Z RE
                         ; (ST,ST(1)) := e**(ST,ST(1))
CALL MOEREXP
FSTP EXP RE
                         ; Store real result, pop ST
      EXP IM
                         ; store imaginary result,
FSTP
                          : pop ST.
: EXP RE is about -2.6911.
; EXP IM is about 0.3836.
```

# mqerCLGE

Returns natural logarithm of complex number

#### **Function**

(ST, ST(1)) := Ln(ST, ST(1))  
(Ln\_Re, Ln\_Im) := Ln(z\_re, z\_im) with 
$$-\pi \le \text{Ln_Im} \le +\pi$$

#### Discussion

CLGE returns the principal value of the complex natural logarithm (base e) for the complex number z with real and imaginary components (z re, z im).

The imaginary component of its result falls in the range  $-\pi..+\pi$ . However, Ln\_Im =  $-\pi$  only under the following conditions:

- z\_re < -0 and
- $z_im = -0$

The mathematical definition for Ln(z) is the complex integral:

$$\operatorname{Ln}(z) = \begin{cases} \frac{dt}{t} & \text{dt} \\ \frac{dt}{t} & \text{dt} \end{cases}$$

The path of integration does not pass through the origin nor cross the negative real axis.

Figure 3-17 shows the domain of Ln(z) with points A = (-1,+0), D = (+1,+0), D' = (+1,-0), A' = (-1,-0), and B = (x,+0), C = (x,+0), C' = (x,-0), B' = (x,-0). Figure 3-18 shows the image of the domain under Ln(z).

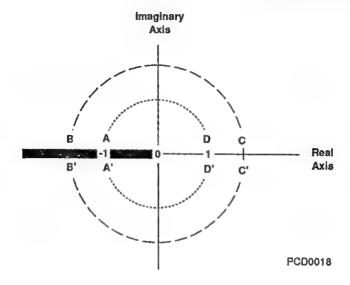


Figure 3-17 Domain of Ln(z) with Branch Cut

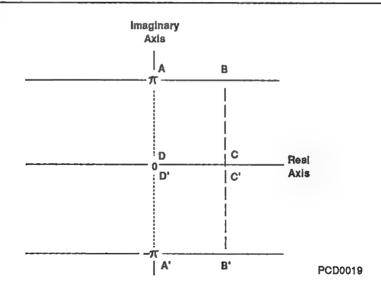


Figure 3-18 Range of Ln(z)

# mqerCLGE (continued) Exceptions

Possible exceptions are Invalid, Denormal, and Zerodivide:

- I Input SNaNs cause the I exception. If I is masked, mqerCLGE returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCLGE returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- Z Input z = (±0, ±0) causes the Z exception. If Z is masked, mqerCLGE returns -∞ to the real component of the result. If Z is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# **Exception Opcode**

397H

## Example

mqerCLGE returns the principal value of Ln(z) for the complex number z.

```
:
                        ; in DATA segment
                      ; initialized
Z RE DT 1.0
Z IM DT 3.0
                        : to test values
LN RE DT ?
LN IM DT ?
                         : in CODE32 segment
EXTRN MOERCLGE: NEAR
: The following lines compute the complex logarithm
; (base e) of the complex number (Z RE, Z IM) and
; store the real and imaginary components
; of the result in LN RE and LN IM.
FLD Z IM
                         ; push, ST := Z IM
FLD Z RE
                        ; push, ST := Z RE
                        ; (ST,ST(1)) := Ln(ST,ST(1))
CALL MQERCLGE
                        ; Store real result, pop ST
FSTP LN RE
                         : Store imaginary result,
FSTP LN IM
                         : pop ST.
; LN RE is about 1.1513.
: LN IM is about 1.2490.
```

### mqerCMUL Returns product of two complex numbers

#### **Function**

#### Discussion

mqerCMUL returns the complex product of two complex numbers w and z with real and imaginary components (w\_re, w\_im) and (z\_re, z im).

The mathematical definition of the complex product is as follows:

$$w * z = (Prod_Re, Prod_Im)$$

where

$$Prod_!m = (w_re * z_im) + (w_im * z_re)$$

For example, 
$$(0, 1) * (0, 1) = (-1, 0)$$
.

The input w and/or z can have one or more infinity components. mqerCMUL calculates results as follows:

- If both components of an argument are finite, mqerCMUL leaves them unchanged.
- It replaces an argument's infinite components with 1 of the same sign and any finite component of this argument with 0 of the same sign.
- Then, it computes the product and replaces each nonzero component of the result with an infinity of the same sign.

# **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- Input SNaNs cause the I exception. If I is masked, mqerCMUL returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCMUL returns its result. If D is unmasked, control passes to the exception handler with w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mgerCMUL returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCMUL returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with w\_im still in ST(3), w\_re still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

## **Exception Opcode**

58CH

# mqerCMUL (continued) Example

mqerCMUL returns the product of two complex numbers w and z.

```
; in DATA segment
                         ; initialized
W RE DT 1.0
W IM DT 2.0
                         : to.
                         ; test
Z_RE DT -3.0
Z IM DT 4.5
                         : values
PROD RE DT ?
PROD IM DT ?
                         : in CODE32 segment
EXTRN MOERCMUL: NEAR
; The following lines compute
; (W RE, W IM) * (Z RE, Z IM) and
; store the real and imaginary components
; of the result in PROD RE and PROD IM.
FLD W IM
                         ; push, ST := W IM
FLD W RE
                         ; push, ST := W RE
FLD Z IM
                         ; push, ST := Z IM
FLD Z RE
                         ; push, ST := Z RE
CALL MOERCMUL
                         ; (ST,ST(1)) := (ST(2),ST(3))
                                      * (ST,ST(1))
                         ; Store real component, pop ST
FSTP PROD RE
                         ; Store imaginary component.
FSTP PROD IM
                         : pop ST.
; PROD RE = 12.0
; PROD IM = -1.5
```

# mgerCPOL

Returns complex number in polar representation for complex number in rectangular representation



#### Discussion

mqerCPOL converts a complex number expressed in rectangular form into the same number expressed in polar form.

mqerCPOL accepts  $z = \pm 0$  and  $z = \pm 0$  returns:

- (+0, 0 with z im's sign) if z re = +0 and z im =  $\pm 0$
- (+0,  $\pi$  with z im's sign) if z re = -0 and z im =  $\pm 0$

mqerCPOL also accepts infinite arguments and returns:

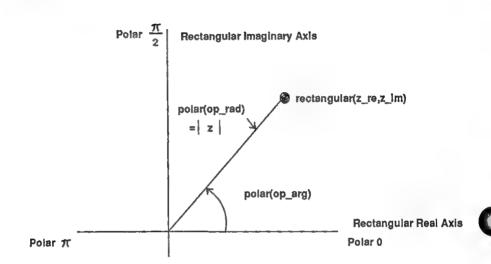
- $(+\infty, 0 \text{ with z im's sign})$  if z re =  $+\infty$  and |z| im  $|< \pm \infty|$
- $(+\infty, \pi/4 \text{ with z_im's sign})$  if z\_re =  $+\infty$  and z\_im =  $\pm\infty$
- $(+\infty, \pi \text{ with z im's sign})$  if z re =  $-\infty$  and |z| im  $|< \pm \infty|$
- $(+\infty, 3*\pi/4 \text{ with z_im's sign})$  if  $z_re = -\infty$  and  $z_im = \pm \infty$

The polar representation of z is defined by the following formulas:

Op\_Rad = 
$$z re^2 + z im^2$$

# mgerCPOL (continued)

mqerCPOL returns a pair of real numbers (Op\_Rad, Op\_Arg) as its result. Op\_Rad is the magnitude of the radius: the line segment from the origin to the point (z\_re, z\_im) in the complex plane, as shown in Figure 3-19. Op\_Arg is the principal value of the angle between the positive real axis and the radius. Op\_Arg is always an angle in radians that falls in the range  $-\pi..+\pi$ .



PCD0020

Figure 3-19 Rectangular and Polar Forms of a Complex Number

# **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCPOL returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCPOL returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# mqerCPOL (continued)

- O An Op\_Rad whose exponent is too large to fit in extended format cause the O exception. If O is masked, mqerCPOL returns +∞ to Op\_Rad. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCPOL returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

## **Exception Opcode**

3A8H

# mqerCPOL (continued) Example

mqerCPOL converts the rectangular form of a complex number with components (z\_re, z\_im) into its polar form with components (op\_rad, op\_arg).

```
: in DATA segment
Z RE DT 1.0
                        : initialized
                        : to test values
Z IM DT 3.0
OP RAD DT ?
OP ARG DT ?
                         : in CODE32 segment
        :
EXTRN MOERCPOL: NEAR
; The following lines convert (Z RE, Z IM)
; from rectangular to polar form and
; store the result in OP RAD and OP ARG.
FLD Z IM
                         ; push, ST := Z IM
FLD Z RE
                         : push, ST := Z RE
CALL MQERCPOL
                         : (ST,ST(1)) := (|ST,ST(1)|),
                         ; principal value(ST, ST(1))
                         : Store radius, pop ST.
FSTP OP RAD
                         : Store angle, pop ST.
FSTP OP ARG
: OP RAD is about 3.1623.
: OP ARG is about 1.2490.
```

mgerCPRJ

Returns (infinity, zero with sign of imaginary component) for complex number with at least one infinite component



(ST, ST(1)) := 
$$(\infty, 0 \text{ with } sign(ST(1)) \text{ if } (ST \text{ or } ST(1) = \infty)$$
  
(Z\_Re, Z\_Im) :=  $(\infty, 0 \text{ with } sign(z_im)) \text{ if } (z_re \text{ or } z_im) = \infty$   
Otherwise, (Z\_Re, Z\_Im) :=  $(z_re, z_im) \text{ unchanged}$ 

#### Discussion

mqerCPRJ returns the complex value ( $\infty$ , 0 with z\_im's sign) if at least one of z's components is infinite. If neither component is infinite, mqerCPRJ returns z unchanged.

mqerCPRJ forms the projective closure of the complex plane by mapping the square at infinity to the single point  $(\infty, \pm 0)$ . The square at infinity consists of all points whose representation in rectangular coordinates has at least one infinite component.

Use mqerCPRJ whenever complex infinities might distort the value expected from a certain expression. In other words, use mqerCPRJ when an expression cannot produce the expected result unless the single point at infinity, as on the Riemann sphere, is the only infinity allowed to occur.

Replace the logical expression  $z = (\infty, 0)$  with mqerCPRJ(z) =  $(\infty, 0)$  whenever you want to detect a complex infinity. Similarly, replace arithmetic expressions of the form  $(w \pm z)$  with expressions of the form  $(\text{mqerCPRJ}(w) - \text{mqerCPRJ}(\mp z))$  only if both w and z might be infinite points other than  $(\infty, \pm 0)$ . Consider using mqerCPRJ only for algebraic functions, like subtraction, that are discontinuous on the Riemann sphere.

# **Exceptions**

See Section 3.1.2.1

# mqerCPRJ (continued) Exception Opcode

3A7H

# Example

mqerCPRJ returns the complex value ( $\infty$ , 0 with the sign of the input imaginary component) if at least one of the input components is an infinity.

```
: in DATA segment
CPRJ RE DT ?
CPRJ IM DT ?
                      ; in CODE32 segment
EXTRN MQERCPRJ: NEAR
; The following lines compute the complex projection
; to infinity of the pair (Z RE, Z IM) and
; store the real and imaginary components
; of the result in CPRJ RE and CPRJ IM.
                       ; push, ST := Z IM
FLD Z IM
                       ; push, ST := Z RE
FLD Z RE
                      ; (ST,ST(1)) := (infinity, 0
CALL MQERCPRJ
                                  with Z IM's sign)
                      ; Store infinite or z re
FSTP CPRJ RE
                      ; component, pop ST.
                       : Store signed zero or z im
FSTP CPRJ IM
                       : component, pop ST.
; CPRJ RE = 7FFF80000000000000 = +infinity
: CPRJ^{T}IM = -0
```

Returns complex result of real raised to complex power

## Function

$$(ST, ST(1)) := ST(2)^{(ST, ST(1))}$$
  
 $(Re\_Result, Im\_Result) := x^{(z\_re, z\_im)} = x^{z}$ 

#### Discussion

mqerCR2C returns the complex result of a real number x raised to the power of the complex number z with real and imaginary components (z re, z im).

mgerCR2C returns:

- $e^{z*in((x, 0))}$  if z im  $\neq \pm 0$
- $(x^{z_{-}r_{e}}, 0 \text{ with } z_{-}r_{e}$ 's sign) if  $x \ge 0$ ,  $z_{-}r_{e}$  is not a finite integer, and  $z_{-}$  im =  $\pm 0$
- e(z\_re\*ln(| x|), (z\_re mod 2)\*principal\_value\_angle(x,0))
   if x not ≥ 0, z\_re is not a finite integer, and z\_im = ±0

The function principal\_value\_angle returns the angle from the positive real axis to the line segment connecting the origin to the point (x, 0); this angle is 0 or  $\pi$  (see mqerCPOL).

For  $z = (z re, \pm 0)$  when z re is a finite integer, mqerCR2C returns:

- $(+1, z_re)$  if  $z_re = \pm 0$
- $(x^{z_{-}r_{e}}, z_{r_{e}})$  with x's sign)\*(1 with  $x^{z_{-}r_{e}}$  sign)) if  $|z_{r_{e}}| > 8$
- $(x, 0)^{z_{-re}}$  if  $1 \le z_{-re} \le 8$  by successively squaring and multiplying (x, 0) to compute the correct power
- (x, 0)<sup>z\_re</sup> if -8 ≤ z\_re ≤ -1 by evaluating one of two possible expressions:
  - 1 / (x, 0) | z\_re | provided that the denominator would not cause numeric overflow or underflow
  - $(1/(x, 0))^{|z|^{re}}$  otherwise

# mqerCR2C (continued) Exceptions

Possible exceptions are Invalid, Denormal, Zerodivide, Overflow, and Underflow:

- I Input SNaNs cause the I exception. If I is masked, mqerCR2C returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with 0 in ST(3), the input x still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCR2C returns its result. If D is unmasked, control passes to the exception handler with 0 in ST(3), x still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- Z Input z\_re < 0 with x and z\_im = ±0 causes the Z exception. If Z is masked, mqerCR2C returns ±∞ to affected components of the result. If Z is unmasked, control passes to the exception handler with 0 in ST(3), x still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCR2C returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with 0 in ST(3), x still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCR2C returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with 0 in ST(3), x still in ST(2), z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# **Exception Opcode**

591H

## Example

mqerCR2C returns the complex result of raising a real number to the power of a complex number.

```
: in DATA segment
X DT 2.0
                         : initialized
7 RE DT 2.0
                         : to test
7 IM DT 0.5
                         : values
RE RESULT DT ?
IM RESULT DT ?
                         : in CODE32 segment
EXTRN MOERCR2C: NEAR
: The following lines compute X**(Z RE.Z IM) and
: return the real and imaginary components of
; the result to RE RESULT and IM RESULT.
                         ; push, ST := X
FLD X
                         : push. ST := Z IM
FLD Z IM
FLD Z RE
                         : push. ST := Z RE
                         : (ST.ST(1)) :=
CALL MOERCR2C
                                     ST(2)**(ST,ST(1))
                        ; Store real component, pop ST
FSTP RE RESULT
FSTP IM RESULT
                         : Store imaginary component,
                         ; pop ST.
: RE RESULT is about 3.7622.
: IM RESULT is about 1.3587.
```

# mgerCREC

Returns complex number in rectangular representation for complex number in polar representation

#### Function

#### Discussion

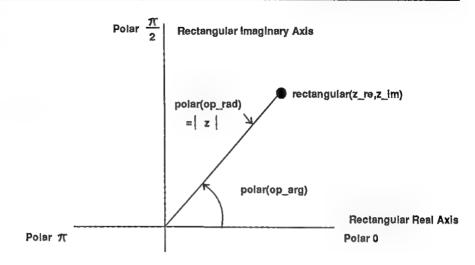
mqerCREC converts a complex number expressed in polar form into the same complex number expressed in rectangular form.

mqerCREC accepts or rad = +\infty and returns:

- $(+\infty, 0 \text{ with op\_arg's sign})$  if  $-\pi/4 < \text{op\_arg} < \pi/4$
- $(+\infty, \pi/4 \text{ with op\_arg's sign})$  if op\_arg =  $\pm \pi/4$
- (+0,  $\infty$  with op\_arg's sign) if  $\pi/4 < |$  op\_arg |  $< 3*\pi/4$
- $(+\infty, 3*\pi/4 \text{ with op\_arg's sign})$  if op\_arg =  $\pm 3*\pi/4$
- $(-\infty, 0 \text{ with op\_arg's sign}) \text{ if } 3*\pi/4 < | \text{ op\_arg} | \leq \pi$

The rectangular representation of z is defined by the following formulas:

The polar representation of the complex number is of the form (op\_rad, op\_arg), as shown in Figure 3-20. Op\_rad is the magnitude of the radius, the line segment from the origin to z; it should not be less than zero. Op\_arg is the angle between the positive real axis and the radius; op\_arg is expressed in radians.



PCD0020

Figure 3-20 Rectangular and Polar Forms of a Complex Number

## **Exceptions**

Possible exceptions are Invalid, Denormal, and Underflow:

- I Input SNaNs, op\_rad < ±0, and op\_arg = ±∞ cause the I exception. If I is masked, mqerCREC returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input op\_arg still in ST(1), op\_rad still in ST, and the exception opcode set.</p>
- D Input denormals cause the D exception. If D is masked, mqerCREC returns its result. If D is unmasked, control passes to the exception handler with op\_arg still in ST(1), op\_rad still in ST, and the exception opcode set.

# mgerCREC (continued)

U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCREC returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with op\_arg still in ST(1), op\_rad still in ST, and the exception opcode set.

## **Exception Opcode**

3A9H

## Example

mqerCREC converts the polar form of a complex number with components (op\_rad, op\_arg) into its rectangular form with components (Z Re, Z Im).

```
; in DATA segment
OP RAD DT 1.0 ; initialized
OP ARG DT 3.0
                        : to test values
Z RE DT ?
Z IM DT ?
                         : in CODE32 segment
        .
EXTRN MQERCREC: NEAR
: The following lines convert a complex number
; in polar form (OP RAD, OP ARG)
: to the number in rectangular form (Z RE, Z IM).
FLD OP ARG
                         ; push, ST := OP ARG
                         : push, ST := OP RAD
FLD OP RAD
CALL MQERCREC
                         ; (ST, ST(1)) := (ST *
                         : cos(ST(1)), ST * sin(ST(1)))
                         : Store real component, pop ST
FSTP Z RE
                         : Store imaginary component,
FSTP Z IM
                         ; pop ST.
: Z RE is about -0.98999.
```

: Z IM is about 0.14112.

## **Function**

(ST, ST(1)) := 
$$sin(ST, ST(1))$$
  
(Sin\_Re, Sin\_Im) :=  $sin(z_re, z_im)$  if  $-\infty < z_re < +\infty$ 

#### Discussion

mqerCSIN returns the complex sine of the complex number z with real and imaginary components (z re, z im).

Sin(z) is defined mathematically in terms of the transcendental constant e (2.71828182845904523536..):

$$\sin(z) = \frac{e^{iz} - e^{-iz}}{2i}$$

The real and imaginary components are defined in terms of real functions:

$$sin(z) = sin(z_re, z_im) = (Sin_Re, Sin_Im)$$

where

$$Sin_Re = sin(z_re) * cosh(z_im)$$

$$Sin Im = cos(z re) * sinh(z im)$$

# mqerCSIN (continued) Exceptions

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs and z\_re = ±∞ cause the I exception. If I is masked, mqerCSIN returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCSIN returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCSIN returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCSIN returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# **Exception Opcode**

399H

## Example

mqerCSIN returns sin(z\_re, z\_im) for the complex number z..

```
; in DATA segment
                        ; initialized
Z RE DT 1.0
Z IM DT 3.0
                         : to test values
SIN RE DT ?
SIN IM DT ?
                         ; in CODE32 segment
EXTRN MQERCSIN: NEAR
; The following lines compute sin(Z RE, Z IM) and
; store the real and imaginary components
; of the result in SIN RE and SIN IM.
FLD Z IM
                         ; push, ST := Z IM
FLD Z RE
                         : push, ST := Z RE
CALL MOERCSIN
                         : (ST,ST(1)) := sin(ST,ST(1))
FSTP SIN RE
                         ; Store real component, pop ST
FSTP SIN IM
                         : Store imaginary component.
                         ; pop ST.
; SIN RE is about 8.4716.
; SIN IM is about 5.4127.
```

# mgerCSNH

Returns complex hyperbolic sine

#### **Function**

(ST, ST(1)) := 
$$sinh(ST, ST(1))$$
  
(Sinh\_Re, Sinh\_Im) :=  $sinh(z_re, z_im)$  if  $-\infty < z_im < +\infty$ 

#### Discussion

mqerCSNH returns the complex hyperbolic sine of the complex number z with real and imaginary components (z\_re, z\_im).

Sinh(z) is defined mathematically in terms of the transcendental constant e (2.71828182845904523536..):

$$\sinh(z) = \frac{e^z - e^{-z}}{2}$$

The real and imaginary components are defined in terms of real functions:

$$sinh(z) = sinh(z_re, z_lm) = (Sinh_Re, Sinh_lm)$$

where

$$Sinh_Re = sinh(z_re) * cos(z_im)$$

$$Sinh_lm = cosh(z_re) * sin(z_im)$$

# **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs and z\_im = ±∞ cause the I exception. If I is masked, mqerCSNH returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCSNH returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCSNH returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCSNH returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# **Exception Opcode**

39AH

# mqerCSNH (continued) Example

mqerCSH returns sinh(z\_re, z\_im) for the complex number z.

```
; in DATA segment
Z RE DT 1.0
                       ; initialized
Z IM DT 3.0
                       : to test values
SINH RE DT ?
SINH IM DT ?
                        ; in CODE32 segment
EXTRN MQERCSNH: NEAR
; The following lines compute sinh(Z RE, Z IM) and
; store the real and imaginary components
; of the result in SINH RE and SINH IM.
                         ; push, ST := Z IM
FLD Z IM
FLD Z RE
                         ; push, ST := Z RE
                         ; (ST,ST(1)) := sinh(ST,ST(1))
CALL MQERCSNH
                        ; Store real component, pop ST
FSTP SINH RE
FSTP SINH IM
                         : Store imaginary component,
                         ; pop ST.
; SINH RE is about -1.1634.
: SINH IM is about 0.2178.
```



#### Discussion

mqerCSQR returns the principal complex square root of the complex number z with real and imaginary components (z\_re, z\_im).

mqerCSQR accepts certain negative and zero input components and returns:

- $(+0, Sqrt_Im > 0)$  if  $z_re < 0$  and  $z_im = +0$
- $(+0, Sqrt_Im < 0)$  if  $z_re < 0$  and  $z_im = -0$
- (+0, 0 with z\_im's sign) if  $z_re = \pm 0$  and  $z_im = \pm 0$

mqerCSQR also accepts component infinities and returns:

- $(+\infty, z \text{ im})$  if  $z \text{ re} = \pm \infty$  and  $z \text{ im} = \pm \infty$
- $(+\infty, z_im)$  if  $z_re \neq \pm \infty$  and  $z_im = \pm \infty$
- $(+\infty, 0 \text{ with z im's sign})$  if z re =  $+\infty$  and z im  $\neq \pm \infty$
- (+0,  $\infty$  with z\_im's sign) if z\_re =  $-\infty$  and z\_im  $\neq \pm \infty$

# **Exceptions**

Possible exceptions are Invalid and Denormal:

I Input SNaNs cause the I exception. If I is masked, mqerCSQR returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# mgerCSQR (continued)

D Input denormals cause the D exception. If D is masked, mqerCSQR returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# **Exception Opcode**

396H

## Example

mqerCSQR returns the principal complex square root of the complex number z.

```
: in DATA segment
                        ; initialized
Z RE DT 1.0
                        : to test values
Z IM DT 3.0
SORT RE DT ?
SORT IM DT ?
                         ; in CODE32 segment
EXTRN MOERCSOR: NEAR
: The following lines compute the complex square root
; of the complex number (Z RE, Z IM) and
: store the real and imaginary components
; of the result in SQRT RE and SQRT IM.
                         ; push, ST := Z IM
FLD Z IM
                         : push, ST := Z RE
FLD Z RE
CALL MQERCSQR
                         ; (ST,ST(1)) := square root(
                                      ST.ST(1))
                         ; Store real component, pop ST
FSTP SORT RE
                         ; Store imaginary component.
FSTP SQRT IM
                         : pop ST.
: SORT RE is about 1.4426.
; SQRT IM is about 1.0398.
```

#### **Function**

(ST, ST(1)) := 
$$tan(ST, ST(1))$$
  
(Tan\_Re, Tan\_Im) :=  $tan(z_re, z_im)$  if  $-\infty < z_re < +\infty$ 

#### Discussion

mqerCTAN returns the complex tangent of the complex number z with real and imaginary components (z\_re, z\_im).

Tan(z) is defined mathematically in terms of sin(z) and cos(z):

$$tan(z) = \frac{\sin(z)}{-\cos(z)}$$

The real and imaginary components are defined in terms of real functions:

$$tan(z) = tan(z_re, z_im) = (Tan_Re, Tan_im)$$

where

$$tan(z_re)$$

$$Tan_Re = \frac{cosh^2(z_im) + (tar^2(z_re) * sinh^2(z_im))}{cosh^2(z_im) + (tar^2(z_re) * sinh^2(z_im))}$$

$$Tan\_Im = \frac{(1 + tar^2(z\_re)) * (sinh(z\_im) * cosh(z\_im))}{cosh^2(z\_im) + (tar^2(z\_re) * sinh^2(z\_im))}$$

# mqerCTAN (continued) Exceptions

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs and z\_re = ±∞ cause the I exception. If I is masked, mqerCTAN returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCTAN returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCTAN returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z im still in ST(1), z re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCTAN returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

# **Exception Opcode**

39DH

## Example

mgerCTAN returns tan(z re, z im) for the complex number z.

```
: in DATA segment
                        ; initialized
Z RE DT 1.0
Z IM DT 3.0 .
                        : to test values
TAN RE DT ?
TAN IM DT ?
                         ; in CODE32 segment
EXTRN MQERCTAN: NEAR
: The following lines compute tan(Z RE, Z IM) and
; store the real and imaginary components
; of the result in TAN RE and TAN IM.
                         ; push, ST := Z IM
FLD Z IM
FLD Z RE
                         : push, ST := Z RE
CALL MQERCTAN
                         ; (ST,ST(1)) := tan(ST,ST(1))
                         ; Store real component, pop ST
FSTP TAN RE
                         ; Store imaginary component,
FSTP TAN IM
                         : pop ST.
; TAN RE is about 0.0045.
; TAN IM is about 1.0021.
```

# mgerCTNH

Returns complex hyperbolic tangent

#### **Function**

(ST, ST(1)) := 
$$tanh(ST, ST(1))$$
  
(Tanh\_Re, Tanh\_Im) :=  $tanh(z_re, z_im)$  if  $-\infty < z_im < +\infty$ 

### Discussion

mqerCTNH returns the complex hyperbolic tangent of the complex number z with real and imaginary components (z\_re, z\_im).

Tanh(z) is defined mathematically in terms of sinh(z) and cosh(z):

$$tanh(z) = \frac{sinh(z)}{cosh(z)}$$

The real and imaginary components are defined in terms of real functions:

where

Tanh\_Re = 
$$\frac{(\sinh(z_re) * \cosh(z_re)) * (1 + \tan^2(z_im))}{\cosh^2(z_re) + (\sinh^2(z_re) * \tan^2(z_im))}$$

$$tan(z_im)$$

$$Tanh_im = \frac{tan(z_im)}{cosh^2(z_re) + (sinh^2(z_re) * tan^2(z_im))}$$

## **Exceptions**

Possible exceptions are Invalid, Denormal, Overflow, and Underflow:

- I Input SNaNs and z\_im = ±∞ cause the I exception. If I is masked, mqerCTNH returns the QNaN indefinite to affected components of the result. If I is unmasked, control passes to the exception handler with the input z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- D Input denormals cause the D exception. If D is masked, mqerCTNH returns its result. If D is unmasked, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.
- O Component results whose exponents are too large to fit in extended format cause the O exception. If O is masked, mqerCTNH returns ±∞ to affected components of the result. If O is unmasked, control passes to the exception handler with z\_im still in ST(1), z re still in ST, and the exception opcode set.
- U Component results that are too tiny to fit accurately in extended format cause the U exception if U is masked. In this case, mqerCTNH returns a gradual underflow denormal to affected components of the result, if possible; otherwise, it returns ±0. Component results that are too tiny to be represented as normalized numbers cause the U exception if U is unmasked. In this case, control passes to the exception handler with z\_im still in ST(1), z\_re still in ST, and the exception opcode set.

## **Exception Opcode**

39EH

# mqerCTNH (continued) Example

mqerTNH returns tanh(z\_re, z\_im) for the complex number z.

```
: in DATA segment
                      initialized
Z RE DT 1.0
Z IM DT 3.0
                       : to test values
TANH RE DT ?
TANH IM DT ?
                         ; in CODE32 segment
      MQERCTNH: NEAR
EXTRN
; The following lines compute tanh(Z RE, Z IM) and
; store the real and imaginary components
; of the result in TANH RE and TANH IM.
                         ; push, ST := Z IM
FLD Z IM
FLD Z RE
                         : push. ST := Z RE
                         : (ST,ST(1)) := tanh(ST,ST(1))
CALL MOERCTNH
FSTP TANH RE
                        : Store real component, pop ST
                         : Store imaginary component,
FSTP
     TANH IM
                         ; pop ST.
: TANH RE is about 0.7680.
: TANH IM is about -0.0592.
```





# **Contents**

	Ch	apter 4 Exception Handling Library	
٠	4.1	Library Overview	4-1
ø		4.1.1 Declaring EH387 Routines in ASM386 Exception Handl	
		4.1.2 Linking with EH387	
		4.1.3 EH387 Stack Requirements	
		4.1.4 EH387 Register Usage	
		4.1.5 EH387 Parameters and Results	
		4.1.6 ESTATE387	
		4.1.7 Protocols for Writing an EH387 Exception Handler	4-10
		4.1.8 EH387 Exception Handler Template in ASM386	
	4.2	EH387 Reference	
		CODE	
		CODE	
	EN	ODE	4-16
	Fig	ures	
	4-1	ESTATE387 Layout	4-5
	4-2	ARGUMENT Byte in ESTATE387	4-6
1		RESULT Byte in ESTATE387	
	Tab	oles	
	4-1	Atyp and Rtyp Field Values	4-7
	4-2	FORMAT Byte Values	4-9



This chapter has two major sections:

- 1. An overview of the 80387 Exception Handling Library (EH387)
- 2. A reference for both EH387 routines

# 4.1 Library Overview

The 80386-based 80387 Exception Handling Library consists of two utility routines: DECODE and ENCODE. These routines make writing exception handlers easier. Exception handlers are system-level routines called when a floating-point operation reports an unmasked 80387 exception.

A customized exception handler can make your code more efficient, particularly when a program must test for exceptions that seldom occur. For example, suppose you have a program with a loop where overflow might happen once in a thousand iterations. Perhaps the program masks O in the 80387 Control Word and tests the 80387 Status Word's exception byte for a set O bit at every iteration. Then, the program jumps to the next iteration when O is clear but jumps to some corrective code when O is set. For such a program, 999 of 1000 overflow checks would be wasted effort that slowed execution. With DECODE and ENCODE, your program can unmask overflow, put the corrective code in an exception handler, and pay the price for overflow only when it occurs.

DECODE identifies the DC387 routine, CL387 function, or 80387 instruction where an unmasked exception occurred. It also saves the machine state of the 80387 and preserves the offending operation's arguments or results. DECODE eliminates much of the effort needed to determine what operation and which unmasked exception caused the call to an exception handler.

Your exception handler can determine how each unmasked exception should be corrected, according to the needs of your code.

ENCODE performs a choice of concluding actions inside your exception handler, either retrying the offending operation or returning a result specified by the handler. It also restores the saved machine state of the 80387, providing a common path for exiting the exception handler and resuming execution of the interrupted program.

The following subsections explain:

- The declaration of EH387 routines in ASM386 exception handlers
- EH387 linkage requirements, stack requirements, and register usage
- EH387 parameters, results, and the ESTATE387 data structure that contains information stored by DECODE and accessed by the handler and ENCODE
- The protocols for writing an exception handler with EH387
- An ASM386 template for an exception handler

# 4.1.1 Declaring EH387 Routines in ASM386 Exception Handlers

The 80387 Exception Handling Library can be linked to code that is within the same code segment or to code that is in another code segment. However, you must declare the EH387 routines within the calling modules. For example, when a module calls DECODE, make one of the following declarations:

• If you want your exception handler to be within the same segment as EH387, use the EH387N.LIB (near library). Declare DECODE as follows before calling this routine:

CODE32 SEGMENT ER ; Segment name must be CODE32.

; EH387 segments are USE32.

EXTRN DECODE: NEAR ; DECODE is now callable.

• If you want your exception handler to be in a different segment from EH387, use the EH387F.LIB (far library). Declare DECODE as follows before calling this routine:

EXTRN DECODE: FAR ; Declaration must be outside

; all SEGMENT..ENDS pairs

; of the module.

See Section 4.1.8 for an example that declares DECODE and ENCODE for linkage with the EH387 near library. See the reference pages in Section 4.2 for ASM386 examples of how to declare these routines for linkage with the EH387 far library. See Appendix B for more information about declaring these routines in high-level language modules.

## 4.1.2 Linking with EH387

EH387 can be linked with the OMF386 output of any Intel translator to execute on an 80387 or on an 80386 with a true software emulator.

In addition to DECODE and ENCODE, EH387 contains a set of alternate PUBLIC names for its own routines, which are used by some Intel translators. See Appendix A for a list of all DC387, CL387, and EH387 PUBLIC symbols.

## 4.1.3 EH387 Stack Requirements

EH387 requires no working 80387 stack positions. EH387N.LIB requires 384 bytes on the 80386 stack for its internal storage; EH387F.LIB requires 448 bytes. EH387 itself allocates the required stack bytes within its modules. However, a reentrant exception handler should allocate an additional 384 (near library) or 448 (far library) bytes on the 80386 stack at each recursive call.

## 4.1.4 EH387 Register Usage

EH387 conforms to the register-saving conventions of Intel 80386 high-level languages. According to these conventions:

- The EH387 routines must leave the 80386 DS, ES, SS, and EBP registers unchanged. For EH387N.LIB, ES and SS are assumed to access the same combined data/stack segment (DATA) as DS.
- The EH387 routines may destroy the contents of the EAX, EBX, ECX, EDX, EDI, ESI, FS, and GS registers.

However, an exception handler is an interrupt routine. It must save the 80386 registers whose contents it alters during execution. See Section 4.1.7 for more information about exception handler protocols.

#### 4.1.5 EH387 Parameters and Results

Parameters to the EH387 routines must be pushed onto the 80386 stack. The segment part of a far pointer must be pushed as a 4-byte quantity because EH387 uses a 32-bit wide stack. Any push of a segment register in a USE32 segment automatically pushes 4 bytes on the stack. Both DECODE and ENCODE pop their parameters from the stack before returning control to the exception handler.

DECODE and ENCODE both require the following parameters:

- A pointer to an ESTATE387 data structure
- An uncleared copy of the 80387 exception byte from the interrupted code

ENCODE requires two additional parameters:

- A version of the 80387 Control Word, possibly altered by the exception handler to retry the interrupted operation
- A Boolean byte that the exception handler sets (1) to make ENCODE re-execute the interrupted operation and clears (0) to make ENCODE restore the 80387 machine state

DECODE stores its results, including 80387 machine state information, in the ESTATE387 data structure. ENCODE restores the information saved by DECODE, including any changes to ESTATE387 made in the exception handler, to the 80387 machine state.

See the reference pages in Section 4.2 for details about parameters to these routines and Section 4.1.6 for a description of ESTATE387.

#### 4.1.6 ESTATE387

ESTATE387 is a 158-byte data structure filled by the EH387 DECODE routine. Its fields contain information about the identity of the offending operation, the formats and values of its arguments, and the machine state of the 80387. For certain 80387 numeric exceptions, ESTATE387 contains already calculated results, rather than arguments. Figure 4-1 illustrates the ESTATE387 data structure.

				4
ARG1(0)	ARGUMENT	OPERA	ATION	_
ARG1(2)	AR	G1(1)	ARG1(0)	
ARG1(4)	AR	G1(3)	ARG1(2)	7
ARC	32(0)	ARG1_FULL	ARG1(4)	1
ARC	32(2)	ARC	32(1)	
ARG	32(4)	ARC	32(3)	1
RES	51(0)	RESULT	ARG2_FULL	7
RES	51(2)	RE	S1(1)	
RES	51(4)	RE	\$1(3)	1
RES2(1)	RE	S2(0)	RES1_FULL	
RES2(3)	RE	S2(2)	RES2(1)	7
RES2_FULL	RE	S2(4)	RES2(3)	7
SAV	E387	REGISTER	FORMAT	1
	(80387 State is 108	bytes for USE32)	,	11

The byte offsets in Figure 4-1 are decimal values, as are the offsets mentioned in the following description of each ESTATE387 field:

OPERATION is a 16-bit word at offset 0. It contains an exception

opcode identifying which DC387 routine, CL387 function, or 80387 instruction reported the unmasked exception. See Appendix F for a summary of the DC387 and CL387 exception opcodes. See Appendix E

for the 80387 instruction exception opcodes.

ARGUMENT is a byte at offset 2. It identifies the types and

locations of the interrupted operation's arguments.

Figure 4-2 shows the format of the ARGUMENT byte.

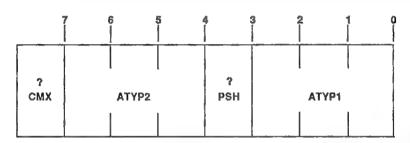


Figure 4-2 ARGUMENT Byte in ESTATE387

PCD0022

PSH (bit 3) is set (1 = true) if the result is to be pushed onto the 80387 stack rather than replacing one or more input arguments.

CMX (bit 7) is set (1 = true) if a CL387 complex function caused the exception: arguments to the offending function are in the ST..ST(1) or ST..ST(3) fields of SAVE387, not in the ESTATE387 ARG1 and ARG2 fields.

ATYP1 and ATYP2 (bits 0..2 and 4..6) indicate the 80387 data types of ARG1 at offset 3 and of ARG2 at offset 14 in ESTATE387, according to the code values in Table 4-1.

Table 4-1 Atyp or Rtyp Field Values

	Table 4-1 Atyp or Ktyp Fleid Values			
_	Value	Meaning		
<b>A</b>	0	no operand		
	1	ST - 80387 stack top		
	2	ST(1) - next 80387 stack element		
	3	ST(i) - element specified by REGISTER field (at offset 49 in ESTATE387)		
	4	number in 80386 memory of type specified by FORMAT field (at offset 48 in ESTATE387)		
	5	80-bit extended format operand		
	6	64-bit long integer operand		
	7	80-bit BCD (binary coded decimal) integer operand		
9	ARG1	is a 5-word array at offset 3 in the ESTATE387 structure. It contains the destination argument (or leftmost argument for CL387 functions) to the operation in the format specified by ARGUMENT's ATYP1 field (see Figure 4-2).		
	ARG1_FULL	is a Boolean byte at offset 13. Its least significant bit is set (1 = true) if ARG1 is present. ARG1 is undefined if the exception handler is called after the interrupted operation is complete.		
	ARG2	is a 5-word array at offset 14. It contains the second argument to the operation in the format specified by ARGUMENT's ATYP2 field (see Figure 4-2).		
	ARG2_FULL	is a Boolean byte at offset 24. Its least significant bit is set if ARG2 is present. ARG2 is undefined if there is only one argument or if the exception handler is called after the interrupted operation is complete.		
•	RESULT	is a byte at offset 25. It identifies the types and locations of the interrupted operation's results. Figure 4-3 shows the format of the RESULT byte.		

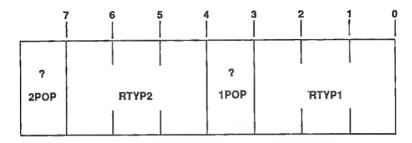


Figure 4-3 RESULT Byte in ESTATE387

PCD0023

1POP (bit 3) is set if the operation causes the 80387 stack to pop exactly once.

2POP (bit 7) is set if the operation causes the 80387 stack to pop twice.

RTYP1 and RTYP2 (bits 0..2 and 4..6) indicate the 80387 data types of RES1 and of RES2, using the code values listed in Table 4-1.

RES1 is a 5-word array at offset 26 in the ESTATE387 structure. It contains the first result of the operation in the format specified by RESULT'S RTYP1 field.

RES1\_FULL is a Boolean byte at offset 36. Its least significant bit is set (1 = true) if RES1 is present. RES1 is undefined if the exception handler is called before the interrupted operation is complete.

RES2 is a 5-word array at offset 37. It contains the second result of the operation in the format specified by RESULT's RTYP2 field.

RES2\_FULL is a Boolean byte at offset 47. Its least significant bit is set if RES2 is present. RES2 is undefined if only one result exists or if the exception handler is called before the interrupted operation is complete.

FORMAT is a byte at offset 48. It specifies the 80387 data type in memory when an ATYP or RTYP field of ARGUMENT or RESULT has the value 4 (see Table 4-1); only one such field ever exists. Table 4-2 shows possible values for FORMAT.

Table 4-2 FORMAT Byte Values

	Table 4-2 TORMAL Dyte Values	
Value	Meaning	
0	32-bit single format	
1	32-bit short integer	
2	64-bit double format	
3	16-bit word integer	
REGISTER	is a byte at offset 49 in the ESTATE387 structure. It specifies the 80387 stack position for an argument or result when an ATYP or RTYP field (see Figures 4-2 and 4-3) has the value 3 (see Table 4-1). The values of REGISTER range from 0 for ST (stack top) to 7.	
SAVE387	is a 108-byte array at offset 50. It contains the machine state of the 80387 as defined by the 80387 FSAVE/FNSAVE instructions in USE32 segments: a 32-bit 80387 Environment followed by the 80387 stack. ST is at offset 78 in ESTATE387.	

The 80387 Status Word stored in SAVE387 is not the interrupted operation's Status Word. The exception handler must clear 80387 exceptions before calling the EH387 DECODE routine, and the handler must maintain the uncleared Status Word separately as a required parameter to both EH387 routines. See Section 4.1.7 for more information about these requirements.

# 4.1.7 Protocols for Writing an EH387 Exception Handler

EH387 eliminates the difficult aspects of interfacing to the 80387 for exception recovery. However, a strict protocol must be followed:

- 1. The handler must be an interrupt routine for the 80386 Processor Extension Error Trap (interrupt vector 16). You must enter its interrupt descriptor in the IDT (Interrupt Descriptor Table) using the system builder (BLD386).
- 2. Preserve all 80386 registers and use the IRETD instruction (see step 9) to transfer control back to the code that called the handler. The 80386 EFLAGS register is automatically saved and restored because the handler is an interrupt routine.
- 3. Use FNSTSW as the first 80387 instruction in the handler. This instruction stores the 80387 Status Word in a 16-bit memory location. The uncleared Status Word from the interrupted operation is a required parameter (XCPTNS387) for both EH387 routines (see steps 5 and 7).
- 4. Clear the 80387 exceptions with an FNCLEX instruction next. Exceptions must be cleared for the EH387 calls to work properly.
- 5. Push the parameters (ESTATE387\_PTR and XCPTNS387) to DECODE onto the 80386 stack and call DECODE.
- 6. Put your customized code for the exception handler next.
- 7. If the handler is returning to the calling environment, push the parameters to ENCODE (ESTATE387\_PTR, XCPTNS387, RETRY\_CONTROL, and RETRY\_FLAG) onto the 80386 stack and call ENCODE.
- 8. Restore the 80386 registers saved at the beginning of the exception handler (see step 2).
- 9. Return control to the interrupted code with an IRETD instruction.

See Section 4.1.8 for an ASM386 template that follows these protocols. Note that a recursive call to the exception handler can occur if ENCODE retries an operation with exceptions unmasked. Allocate all data storage used by DECODE, the handler, and ENCODE on the 80386 stack if recursion is possible.

# 4.1.8 EH387 Exception Handler Template in ASM386

```
NAME HANDLER 387
                         : module name
: DATA segment can have declarations here.
: For EH387N.LIB. DS. ES. and SS point to combined
: data/stack segment named DATA. For EH387F.LIB, DS
: points to DATA and SS points to STACK.
CODE32 SEGMENT ER PUBLIC
I MASK EOU 0001H
                         : I exception is bit 0 in
                         : the 80837 Control Word
                         : and Status Word.
                         ; Masks for other exceptions
    :
                         : can go here.
EXTRN DECODE: NEAR
                         : EH387 routines must be
EXTRN ENCODE: NEAR ; declared external.
: STACK LAYOUT includes ESTATE387. It sets up
; a stack frame for the TRAP HANDLER routine.
STACK LAYOUT STRUC
                      ; pointed at by EBP
 OPERATION DW ?
                        : ESTATE387 begins here.
 ARGUMENT DB ?
 ARG1 DW 5 DUP(?)
 ARG1 FULL DB ?
 ARG2 DW 5 DUP(?)
 ARG2 FULL DB ?
 RESULT DB ?
 RES1 DW 5 DUP(?)
 RES1 FULL DB ?
 RES2 DW 5 DUP(?)
  RES2 FULL DB ?
  FORMAT DB ?
 REGISTER DB ?
            ; Next is the 108-byte SAVE387 array for
            ; 80387 machine state saved in ESTATE387.
 CONTROL WORD DW 2 DUP(?)
 STATUS WORD DW 2 DUP(?)
                                      : to be saved
                         ; after exceptions cleared
 TAG WORD DW 2 DUP(?)
```

```
: XCPTN PTRS includes 32-bit Real Address mode's 80387
; Instruction Pointer, Opcode, and Operand Pointer
; or 32-bit Protected mode's 80387 Instruction Pointer
; Offset, Opcode, CS Selector, Operand Pointer
; Offset, and Operand Selector fields.
  XCPTN PTRS DO 2 DUP(?)
  STACK387 DT 8 DUP(?) : ESTATE387 ends here.
  RETRY CONTROL DW ?
                         ; retry 80387 Control
                         : Word setting for ENCODE
                         : Boolean parameter to ENCODE
  RETRY FLAG DB ?
  XCPTNS387 DW ?
                         : 80387 Status Word
                         : before it is cleared
: Put any additional stack-allocated variables
: for custom code here. Reference them as
: [EBP].vour var name. Additional stack bytes should
; align on a 4-byte boundary. Change the allocation
; for the following dummy variable as necessary.
  FOUR ALIGN DB? : Keep ESP on 4-byte boundary.
; REGISTERS386 is allocation for the 80386 registers
: saved by TRAP HANDLER routine.
  REGISTERS386 DD 12 DUP (?)
  SAVE EIP DD ?
                         : for return from
 SAVE CS DD ?
                       : TRAP HANDLER routine
  FLAGS386 DD ?
                         : for EFLAGS register pushed
                         ; by 80386 when INT call
                         : causes control transfer
                         : to TRAP HANDLER routine
STACK LAYOUT ENDS
; TRAP HANDLER is a template for an 80387 exception
; handler in ASM386 using EH387. It assumes that the
; only unmasked exception is Invalid Operation.
```

TRAP\_HANDLER PROC FAR

: The next five lines save the 80386 general and : segment registers on the stack. You can modify ; these to save only those registers that your ; handler uses. If you make changes here, you should ; make corresponding changes immediately before : TRAP HANDLER's IRETD, and you should alter : REGISTERS386 in the STACK LAYOUT to allocate ; as much storage as your saved registers need. : Push 8 general registers. PUSHAD : Push data segment PUSH DS PUSH ES : registers. PUSH FS PUSH GS : Allocate room for rest of STACK LAYOUT structure. SUB ESP, OFFSET REGISTERS386 : Set up indexing into STACK LAYOUT. MOV EBP, ESP ; Save exceptions that caused interrupt to handler. FNSTSW [EBP].XCPTNS387 : Clear exceptions so exception handler can use 80387. FNCLEX : Push parameters to EH387 DECODE routine onto stack : with PUSH ESTATE XCPTNS routine, declared following : TRAP HANDLER. CALL PUSH ESTATE XCPTNS CALL DECODE : fills ESTATE387 : The next three lines make Control Word of the ; interrupted code the default for an ENCODE retry. MOV AX, [EBP].CONTROL WORD MOV [EBP].RETRY CONTROL. AX MOV [EBP].RETRY FLAG, OFFH ; Retry will

: occur.

- ; Mask I exception: ENCODE retry will supply
- ; default results for Invalid Operation. OR [EBP].RETRY\_CONTROL,I\_MASK
- ; Was I exception bit set?
   TEST [EBP].XCPTNS387, I\_MASK
   JNZ ACTUAL I XCPTN
- ; Examine [EBP].XCPTNS387 here to detect other
- ; exceptions, and insert your customized code
- ; between the labels ACTUAL I XCPTN and ENCODE EXIT
- ; to handle other unmasked exceptions.

JMP ENCODE EXIT

## ACTUAL\_I\_XCPTN:

- ; Put your customized code for I exceptions here.
- ; You may include a call to ENCODE if you want it to
- ; retry the operation that reported the I exception,
- ; but clear RETRY FLAG before exit through ENCODE EXIT
- ; if you don't want another retry.

# ENCODE\_EXIT:

- ; Push first two ENCODE parameters with a call to
- ; PUSH\_ESTATE\_XCPTNS, declared following TRAP\_HANDLER. CALL PUSH\_ESTATE\_XCPTNS
- ; Push third ENCODE parameter. PUSH DWORD PTR [EBP].RETRY\_CONTROL
- ; Push fourth ENCODE parameter. PUSH DWORD PTR [EBP].RETRY\_FLAG
- ; Restore post-exception 80387 State. CALL ENCODE
- ; Release the STACK LAYOUT storage up to REGISTERS386. ADD ESP, OFFSET REGISTERS386

- ; The next five lines restore the 80386 general
- ; and segment registers that were saved at
- ; the beginning of TRAP HANDLER.
  - POP GS
  - POP FS
  - POP ES
  - POP DS
  - **POPAD**
- : Return must be an IRETD instruction if TRAP HANDLER
- ; was called from a USE32 segment with an INT
- ; instruction. IRETD restores 80386 EFLAGS register. IRETD

TRAP HANDLER ENDP

- : PUSH ESTATE XCPTNS pushes two parameters
- ; (ESTATE387 PTR and XCPTNS387) to the EH387
- ; DECODE and ENCODE routines onto the stack.

PUSH ESTATE XCPTNS PROC NEAR

POP EDX ; Save return address.

- ; For EH387F.LIB, segment part of ESTATE387 PTR
- ; should be pushed here with PUSH SS.
- ; For EH387N.LIB, you need to push only
- ; the offset part of ESTATE387 PTR.
- ; Get offset part of ESTATE387\_PTR. LEA EAX, [EBP].OPERATION

**PUSH EAX** 

; ESTATE387\_PTR pushed

; Push 80387 exceptions in low-order byte. PUSH DWORD PTR [EBP].XCPTNS387

JMP EDX
PUSH ESTATE XCPTNS ENDP

; return to TRAP\_HANDLER

# 4.2 EH387 Reference

This section contains a detailed reference for the EH387 DECODE and ENCODE routines.

#### DECODE

Decode trapped operation and save 80387 status

#### **Parameters**

Push these parameters onto the 80386 stack in the following order before calling DECODE:

- 1. ESTATE387\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to an ESTATE387 data structure (see Section 4.1.6). For a far ESTATE387\_PTR, push ESTATE387\_PTR's segment part first (32-bit push), followed by the offset.
- XCPTNS387 is a 16-bit variable pushed as 32 bits. Its low-order byte is the 80387 Status Word's exception byte. XCPTNS387 contains the 80387 exception byte in existence when the exception handler is called, before 80387 exceptions are cleared (see Section 4.1.7). For DECODE, only the low-order byte of XCPTNS387 is meaningful.

#### Discussion

DECODE supplies information in a standardized format to an exception handler. This information includes:

- The exception opcode that identifies the offending 80387 instruction, DC387 routine, or CL387 function (Appendix E lists exception opcodes for instructions, Appendix F for DC387 and CL387 routines.)
- The existence, formats, and values of arguments or results
- The machine state of the 80387 when the exception occurred

DECODE returns this information to the 158-byte buffer accessed by ESTATE387\_PTR (see Section 4.1.6 for a detailed description of the ESTATE387 fields). DECODE leaves the 80387 completely cleared to its initialized state, ready for use by an exception handler and ready for a call to the EH387 ENCODE routine. For 80387 FCOMP/FCOMPP and FUCOMP/FUCOMPP instructions with denormal arguments, the exception handler is called with the inputs already popped off the 80387 stack. DECODE recovers these arguments and pushes them back on the top of the stack.

## Example

This example assumes declaration of the ESTATE387 structure inside STACK\_LAYOUT as in Section 4.1.8, but not the declaration of the PUSH\_ESTATE\_XCPTNS routine.

```
EXTRN DECODE: FAR
                         ; must be outside all
                        : SEGMENT..ENDS in module
STACK LAYOUT STRUC
STACK LAYOUT ENDS
DATA32 SEGMENT
E STATE STACK LAYOUT ; ESTATE387 is first 158 bytes
DATA32 ENDS
HANDLER CODE SEGMENT ER PUBLIC
; Assume that DS points to DATA32 segment and that
; SS points to STACK32. Default for SEGMENT is USE32.
FNSTSW E STATE.XCPTNS387
FNCLEX
PUSH DS
                         : segment(E STATE) onto 80386
                         ; stack (4-byte push)
LEA EAX, E STATE ; EAX := offset(E STATE)
                         ; E STATE PTR offset
PUSH EAX
                         ; onto stack
PUSH DWORD PTR E STATE.XCPTNS387
                         : XCPTNS387 onto stack
CALL DECODE
; E STATE is now filled with information for use
; by the exception handler and ENCODE.
```

### **ENCODE**

Restores 80387 status and operation environment

#### **Parameters**

Push these parameters onto the 80386 stack in the following order before calling ENCODE:

- 1. ESTATE387\_PTR is either a near (32-bit offset) or a far (16-bit segment, 32-bit offset) pointer to the ESTATE387 data structure that was filled by a preceding call to DECODE and possibly modified by the handler. For a far ESTATE387\_PTR, push the segment part first (32-bit push), followed by the offset.
- XCPTNS387 is a 16-bit variable pushed as 32 bits. Its low-order byte is the 80387 Status Word's exception byte. XCPTNS387 contains the 80387 exception byte in existence when the exception handler is called, before 80387 exceptions are cleared (see Section 4.1.7). For ENCODE, only the low-order byte of XCPTNS387 is meaningful.
- 3. RETRY\_CONTROL is a 16-bit variable pushed as 32 bits. It is a copy of the 80387 Control Word that can be modified by the exception handler. If RETRY\_FLAG is false, ENCODE ignores RETRY\_CONTROL. Otherwise, ENCODE substitutes RETRY\_CONTROL for the 80387 Control Word when it retries the operation that caused the original exception.
- 4. RETRY\_FLAG is a one-byte variable pushed as 32 bits. It is a Boolean control input. If RETRY\_FLAG is set (1 = true), ENCODE reads ESTATE387 to identify the operation that caused the exception, to restore the (possibly modified) 80387 machine state, and to retry the operation.

#### Discussion

Before calling ENCODE, the 80387 must be completely cleared to its initialized state. ENCODE obtains information about the 80387 from the ESTATE387 data structure (see Section 4.1.6), not from the entry state of the 80387 itself. ENCODE restores the 80387 machine state as it is represented in ESTATE387.

If the least significant bit of RETRY\_FLAG is 0, ENCODE assumes that the operation has already been performed and that results have been stored in ESTATE387. ENCODE copies the results to their destination, storing an appropriate 80387 indefinite value (see Appendix D) if results are missing from ESTATE387.

If the least significant bit of RETRY\_FLAG is 1, ENCODE identifies the offending operation from ESTATE387 and retries it with the arguments from ESTATE387. However, it uses RETRY\_CONTROL as the 80387 Control Word for this retry operation; thus, the handler determines which exceptions are unmasked. After the retry operation is complete, ENCODE restores the 80387 Control Word from ESTATE387.

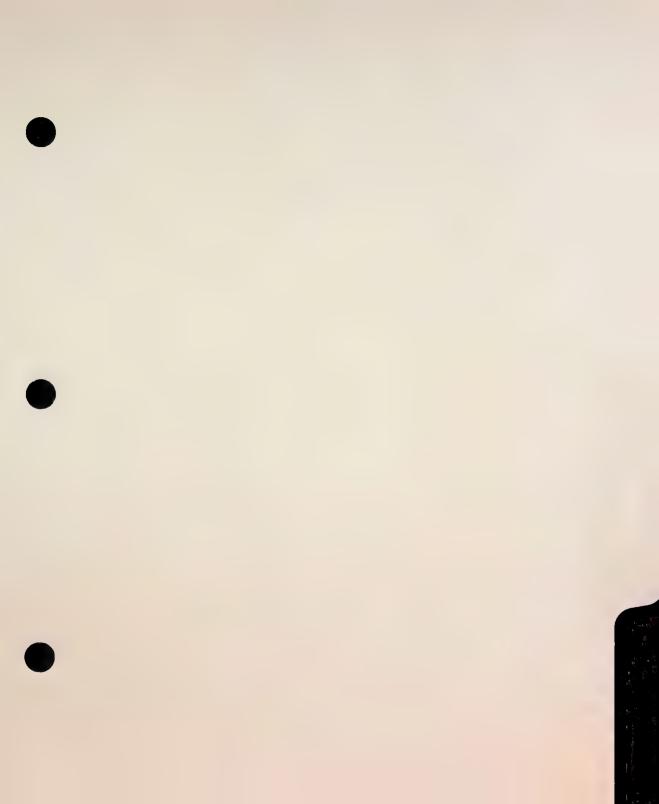
If RETRY\_CONTROL has unmasked exceptions, a recursive call to the exception handler can occur. To avoid infinite recursion, the exception handler must modify the arguments if it unmasks the original exception in RETRY\_CONTROL, sets RETRY\_FLAG, and calls ENCODE.

ENCODE returns with the 80387 restored to the post-exception state determined by the exception handler.

# ENCODE (continued) Example

This example assumes declaration of the ESTATE387 structure inside STACK LAYOUT as in the DECODE example.

```
: must be outside all
EXTRN ENCODE: FAR
                          : SEGMENT.. ENDS in module
STACK LAYOUT STRUC
STACK LAYOUT ENDS
DATA32 SEGMENT
E STATE STACK LAYOUT
                          : ESTATE387 is first 158 bytes
                          : RETRY FLAG, RETRY CONTROL.
                          : XCPTNS387, and FOUR ALIGN
                          : are next 6 bytes. Stack
                          ; should be aligned
                          : on 4-byte boundaries.
DATA32 ENDS
HANDLER CODE SEGMENT ER PUBLIC
; Assume that DS points to DATA32 segment, SS points
: to STACK32, and that parameters to ENCODE are set
: to appropriate values. Default for SEGMENT is USE32.
                          : segment(E STATE) onto 80386
PUSH DS
                          ; stack (4-byte push)
                          ; EAX := offset(E STATE)
LEA EAX, E STATE
                          : E STATE PTR offset
PUSH EAX
                          ; onto stack
MOV AX, E STATE.XCPTNS387
PUSH EAX
                          : XCPTNS387 onto stack
MOV AX, E STATE.RETRY CONTROL
PUSH EAX
                          ; RETRY CONTROL onto stack
MOV AL, E STATE.RETRY FLAG
PUSH EAX
                          ; RETRY FLAG onto stack
CALL ENCODE
: 80387 is now restored to an appropriate
: post-exception state.
```





Ap	pendix A 80387 Support Library PUBLIC Symbols
	Initialization Library PUBLIC SymbolsA-1
	DC387 PUBLIC Symbols
	CL387 PUBLIC Symbols
A.4	EH387 PUBLIC SymbolsA-3
Ap	pendix B High-level Languages and the 80387 Support Library
B.1	Decimal Conversion RoutinesB-1
	B.1.1 Near Call to mqcBIN_DECLOWB-2
	B.1.2 Far Call to mqcDECLOW_BINB-5
B.2	Common Elementary Real FunctionsB-7
	B.2.1 Near Call to mqerACSB-7
	B.2.2 Far Call to mqerIE2B-8
B.3	Exception Handling TemplateB-8
Ap	pendix C ANSI/IEEE Std 754-1985 Conformance
	Decimal Conversion Library
	Common Elementary Function Library
2.3	Exception Handling Library
Ap	pendix D 80387 Numeric Data Formats
	Integer FormatsD-1
D.2	Real FormatsD-2
Ap	pendix E ASM386 Floating-Point Instructions
	pendix F 80387 Support Library Summary
	Parameters and Results for DC387 RoutinesF-1
	Arguments and Results for CL387 FunctionsF-3
F.3	
T A	80387 Exceptions and Exception OpcodesF-8

# **Tables**

C-1	Correctly Rounded Decimal Conversion Ranges Required	C-1
C-2	Decimal Conversion Ranges Required	C-3
D-1	Integer Formats	D-1
	Real Formats	
D-3	Summary of Real Format Parameters	D-3
D-4	Real Zero, Infinity, and QNaN Indefinite Values	D-3
F-1	Summary of CL387 Real Function Arguments and Results	F-4
F-2	Summary of CL387 Complex Function Arguments and Results	.F-5
F-3	Summary of Possible DC387 and CL387 Exceptions	.F-8
F-4	DC387 and CL387 Exception Opcodes	F-10

# Appendix A 80387 Support Library PUBLIC Symbols

Each of the 80387 Support Libraries use some PUBLIC symbols other than the names of routines described in this manual. These are internal names, used either in the libraries or by Intel translators.

The following sections list the PUBLIC names for each 80387 Support Library in alphanumeric order.

# A.1 Initialization Library PUBLIC Symbols

The PUBLIC symbols of 80387N.LIB and 80387F.LIB are INIT87 and INITFP.

# A.2 DC387 PUBLIC Symbols

MQCBINDEC
MQCBIN DECLOW
MQCDBX
MQCDBXDB
MQCDECBIN
MQCDECBINLO
MQCDECLOW BIN
MOCDEC BIN
MQCDUBL XTND
_

MQCHK UNMSKD O U ERR
MQCLONG TEMP
MQCSHORT TEMP
MQCSNGL XTND
MQCSNGXDB
MQCSNX
MQCTEMP LONG
MQCTEMP SHORT
MQCXDB

MQCXDBDB
MQCXDBSNG
MQCXSN
MQCXTND\_DUBL
MQCXTND\_SNGL
MQPOWER\_OF\_10
MQSTACK
MQUNMSKD\_OV\_OR\_UN
MQXCPTN\_RTRN

# A.3 CL387 PUBLIC Symbols

MQERACS MQERAIN MQERASH MQERASH MQERASN MQERAT2 MQERATN MQERCABS MQERCACH MQERCACS MQERCACH MQERCASH MQERCASH MQERCATH MQERCCIC MQERCCIC MQERCCIC MQERCCIC MQERCCIS MQERCCIS MQERCCIS MQERCCIS MQERCCIS MQERCCIS MQERCCIS MQERCCIS MQERCCOS MQERCCSH MQERCLIC MQERCPOL MQERCPOL MQERCPRJ MOERCRCC	MQERIA4 MQERIA8 MQERIC2 MQERIC4 MQERIC8 MQERIC8 MQERIE2 MQERIE4 MQERIE8 MQERIE4 MQERIENT MQERIENT MQERIED MQERLGD MQERLGE MQERMAX MQERMOD MQERMOD MQERNIO MQERNIO MQERNIO MQERNIO MQERRNIO MQERRNIO MQERRNIO MQERSIN	MQ_BLOAT MQ_CASNS MQ_CATNS MQ_CCOSS MQ_CMUL MQ_CONST MQ_COS MQ_CP2N63 MQ_CP2N63 MQ_CTANS MQ_DECIDE MQ_DX1 MQ_DXIT MQ_DXIT MQ_DXPI2 MQ_EXIT MQ_EXIT MQ_IRCHK MQ_LOG MQ_LOGIO MQ_LOGDN MQ_LOGDN MQ_MQRPI MQ_NAN MQ_NOF MQ_NORM MQ_NQ MQ_OF MQ_PO MQ_PI2 MO_PII
MQERCR2C	MQERY2X	MQ_PII
MQERCREC MQERCSH	MQERYI2 MQERYI4	MQ_Q MQ_RAD
MQERCSIN	MQERY18	MQ_RERR
MOERCSNH	MQERYIS	MQ_RSTR_86STAK
MQERCSQR	MQ_1	MQ_SCALCEXP
MQERCTAN	MQ_2XM1	MQ_SIN
MOERCTNH	MQ_63U	MQ_TXAM
MQERDIM	MQ_63U1	MQ_00
MQEREXP	MQ_63UPI2	MQ_YL2X
MQERIA2	MQ_AT2	

# A.4 EH387 PUBLIC Symbols

DECODE ENCODE TQDECODE87 TQENCODE87 TQFETCH\_AND\_STORE TQINSTRUCTION\_RETRY
TQPOP\_THE\_TOP
TQRESTORE\_PTRS
TQSAVE\_PTRS
TOSTACK

TQUNPOP\_THE\_TOP TQ\_320 TQ\_322 TQ\_STATUS\_CHECK



The 80387 Support Library routines can be linked with the OMF386 output of any Intel translator. Thus, high-level language programs can call most 80387 Support Library routines, provided the following requirements are met:

- The routine must be declared external in the calling module.
- Required parameters or arguments to the routine must be set up in the calling language.
- The appropriate near or far 80387 Support Library files must be linked with the calling program.

Some languages, such as C and PL/M, do not support arithmetic operations on complex numbers. Such languages cannot call the CL387 complex functions directly. However, it is possible to write an ASM386 interface routine to push complex arguments onto the 80387 stack, call the CL387 complex function, pop its component results off the 80387 stack, and store them.

This appendix shows PL/M-386 examples of how to declare, call, and use representative Support Library routines as a guide for high-level language programmers. These examples are comparable to the ASM386 examples for mqcBIN\_DECLOW and mqcDECLOW\_BIN in Chapter 2, mqerACS and mqerIE2 in Chapter 3, and the exception handling template in Chapter 4. However, the PL/M calls to INIT\$REAL\$MATH\$UNIT make calls to the 80387 Initialization Library unnecessary.

Consult your language manual for information about its calling conventions.

## **B.1 Decimal Conversion Routines**

The following PL/M-386 examples show a near call to mqcBIN\_DECLOW and a far call to mqcDECLOW\_BIN. The first example must be linked with DC387N.LIB, and the second with DC387F.LIB.

### **B.1.1 Near Call to mqcBIN DECLOW**

```
/*** The PL/M default segmentation model, small ram.
     uses 4-byte (offset only) pointers. For PL/M-386.
     the medium and the small models of segmentation
     use 4-byte pointers, the large and compact models
     6-byte (segment selector and offset) pointers.
 ***/
$WORD32
/*** The WORD32 control is used here. Since the
     WORD32 control changes the size of some data
     types, check the structures if WORD16 is used!
 ***/
BINDEC: DO:
macBIN DECLOW: PROCEDURE (block ptr) EXTERNAL:
   DECLARE block ptr POINTER:
END mgcBIN DECLOW:
DECLARE FALSE LITERALLY '(1=0)':
DECLARE TRUE LITERALLY '(0=0)':
/*** The following procedure converts the WORD value
     from mgcBIN DECLOW to an ASCII equivalent. It
     expects a WORD value and a POINTER value.
     WORD value (a$value) represents a 16-bit integer
     in two's complement notation. The pointer value
     (char$buf$ptr) points to (at least) a 6-byte area
     that will hold an ASCII string. The procedure
     returns the number of characters in the string.
 ***/
int to ascii: PROCEDURE (a$value, char$buf$ptr) BYTE;
   DECLARE char$buf$ptr POINTER:
   DECLARE a$value WORD:
   DECLARE char$buf BASED char$buf$ptr (1) BYTE:
                             BYTE:
   DECLARE count
   DECLARE divisor
                             INTEGER:
   DECLARE index
                             BYTE:
   DECLARE int value
                            INTEGER:
   DECLARE print zeros flag BYTE;
```

```
print zeros flag = FALSE:
   count = 1:
   divisor = 10000:
   int value = INTEGER(a$value):
   IF int value < 0 THEN DO:
       int value = IABS (int value):
       char$buf(0) = '-':
       char$buf$ptr = @char$buf(1):
       count = count + 1:
   END:
   /*** The next loop does work: looping occurs after
        char$buf(0) contains nonzero elements when
        char$buf$ptr is set to address of char$buf(1).
    ***/
   DO index = 1 TO 4:
      char$buf(0) = BYTE(WORD(int value/divisor))+'0':
      int value = int value MOD divisor;
      divisor = divisor / 10:
       IF print zeros flag THEN DO:
           char$buf$ptr = @char$buf(1):
           count = count + 1:
       END:
       ELSE IF char$buf(0) <> '0' THEN DO:
           char$buf$ptr = @char$buf(1):
           count = count + 1:
           print zeros flag = TRUE;
       END:
   END:
   char$buf(0) = BYTE(WORD(int value)) + '0';
   RETURN (count):
END int to ascii:
```

```
/*** ADCB structure is declared next. Pointers to the
     binary and decimal buffers are 4-byte offsets,
     but $buf$ segs fill ADCB fields so ADCB layout
     is proper for mgcBIN DECLOW. b$buf$seq.
     d$buf$seg, and scale are of type HWORD (16 bits)
     because the WORD32 control is used. If the
     WORD16 control is used instead, these fields
     must become WORDs.
***/
DECLARE adcb$block STRUCTURE (
           b$buf$off POINTER.
           b$buf$seq HWORD.
           presn
                    BYTE.
                    BYTE.
           Ingth
           d$buf$off POINTER.
           d$buf$seq HWORD.
           scale
                    HWORD.
           sign
                    BYTE):
DECLARE SNGL PRCSN LITERALLY '0':
DECLARE DUBL PRCSN LITERALLY '2':
DECLARE XTND PRCSN LITERALLY '3':
DECLARE real var3 REAL:
DECLARE digits buffer(6) BYTE:
DECLARE final buffer(15) BYTE:
DECLARE count BYTE:
CALL INIT$REAL$MATH$UNIT:
/* Assume that REAL VAR3 contains the hexadecimal
   value COE9A36D. */
real var3 = REAL
(1100$0000$1110$1001$1010$0011$0110$1101B);
/* C 0
            E
                 9 A 3
                               6
                                     D
adcb$block.b$buf$ptr = @real var3;
adcb$block.prcsn = SNGL PRCSN:
adcb$block.lngth = LENGTH(digits buffer);
adcb$block.d$buf$ptr = @digits buffer;
CALL mgcBIN DECLOW (@adcb$block);
```

## B.1.2 Far Call to mqcDECLOW\_BIN

```
$WORD32
/*** The WORD32 control is used here. Since the
     WORD32 control changes the size of some data
     types, check the structures if WORD16 is used!
***/
$LARGE (NUM387 HAS mgcDECBINLO;
               EXPORTS mgcDECLOW BIN)
$COMPACT
/*** The large subsystem control forces the
      mgcDECLOW BIN call to be a far call (segment
      selector and offset). The rest of the program
      is in the compact model of segmentation; any
      other calls will be near (offset only). The
      compact model of segmentation is used here to
      get 6-byte pointers in the structures.
 ***/
```

```
decbin:DO:
DECLARE SNGL PRSCN LITERALLY '0':
DECLARE adcb STRUCTURE (
   b$buf$ptr POINTER.
             BYTE.
   presn
   Inath
             BYTE.
   d$buf$ptr POINTER.
   scale
             HWORD.
   sian
             BYTE):
mgcDECLOW BIN: PROCEDURE (block ptr) EXTERNAL;
   DECLARE block ptr POINTER;
END mgcDECLOW BIN;
DECLARE real var2 REAL;
DECLARE digits buffer(21) BYTE INITIAL ('371');
CALL INIT$REAL$MATH$UNIT:
/*** The following code assumes DIGIT BUFFER contains
     a decimal string representing an integer number
     of pennies. It writes the number of dollars in
     SNGL PRSCN binary format to REAL VAR2. The
     dollar value to be converted is +371E-2
***/
adcb.b$buf$ptr = @real var2:
adcb.prcsn = SNGL PRSC\overline{N}:
adcb.lngth = 3:
adcb.d$buf$ptr = @digits buffer:
adcb.scale = HWORD(-2):
adcb.sign = '+':
CALL mgcDECLOW BIN(@adcb):
END;
```

## **B.2 Common Elementary Real Functions**

The following PL/M-386 examples show a near call to mqerACS and a far call to mqerIE2. The first example must be linked with CL387N,LIB, and the second with CL387F,LIB.

### **B.2.1 Near Call to mqerACS**

```
acs:DO:
mgerACS: PROCEDURE (x) REAL EXTERNAL;
  DECLARE x REAL:
END mgerACS:
DECLARE hypotenuse REAL:
DECLARE adjacent side REAL:
DECLARE angle radians REAL:
DECLARE angle degrees REAL:
DECLARE PI REAL DATA (3.14159265358979);
CALL INIT$REAL$MATH$UNIT;
/*** The following lines calculate the value of an
     angle of a right triangle, given the length of
     the hypotenuse and the adjacent side. mgerACS
     returns the value in radians: the value is then
     converted to degrees.
***/
hypotenuse = 10.0;
adjacent side = 5.0:
angle radians = mqerACS (adjacent side/hypotenuse);
angle degrees = (180.0/PI) * angle radians;
END:
```

### **B.2.2 Far Call to mqerIE2**

```
$COMPACT
$LARGE (NUM387 HAS RI2;
               EXPORTS mgerIE2)
/*** The large subsystem control forces the mgerIE2
     call to be a far call (segment selector and
     offset).
 ***/
ie2:D0:
mgerIE2: PROCEDURE (x) SHORTINT EXTERNAL;
   DECLARE x REAL:
END mgerIE2:
DECLARE real var REAL:
DECLARE integer var SHORTINT;
CALL INIT$REAL$MATH$UNIT:
real var = -8.5:
integer var = mqerIE2(real var);
/*** integer var is now -8 ***/
END:
```

## **B.3 Exception Handling Template**

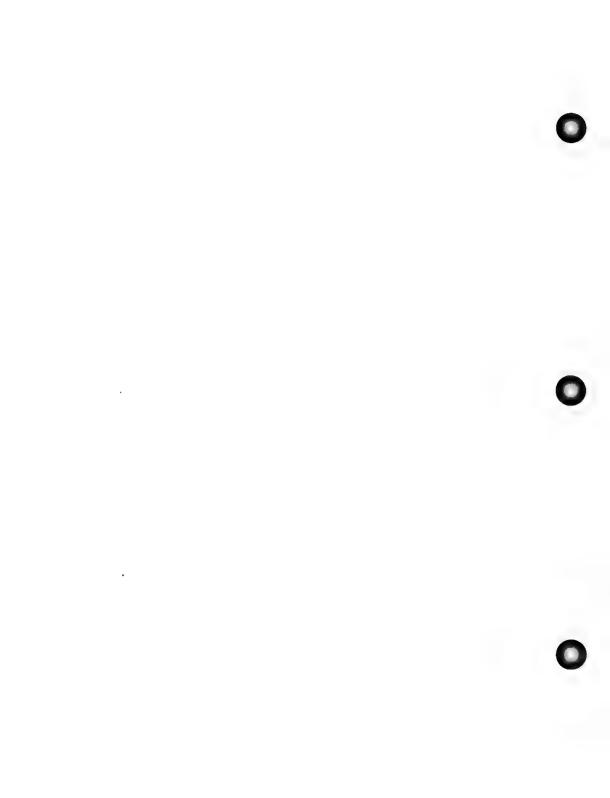
This PL/M-386 template assumes that the 80386 will execute in 32-bit protected mode, as shown by the Env387 structure declaration. It also assumes that the handler module will be linked with EH387N.LIB.

```
$WORD32
handler_module: DO;
```

```
/*** All the HWORDs are 16-bit quantities because
     the WORD32 control is used. (WORDs are 32-bit
     quantities.) If the WORD16 control is used,
     the HWORDs must become WORDs, and the WORDs must
     become DWORDs.
***/
DECLARE I$error$bit LITERALLY '0001H':
/*** I exception is bit 0 in the 80387 Control and
     Status Words. Masks for other exceptions can
     go here.
***/
DECLARE True LITERALLY 'OFFH':
DECLARE Env387 LITERALLY 'STRUCTURE (
         control word
                          HWORD.
         reserved 1
                          HWORD.
         status word
                         HWORD.
   /*** status word after clearing exceptions ***/
         reserved 2
                          HWORD.
         tag word
                          HWORD.
         reserved 3
                          HWORD.
         eip offset
                          WORD.
         cs selector
                          HWORD.
         opcode
                          HWORD.
         data operand off WORD.
         operand selector HWORD,
                          HWORD) ';
         reserved 4
DECLARE Stack387 LITERALLY 'STRUCTURE (
         stack 0(10)
                          BYTE.
         stack 1(10)
                          BYTE.
         stack 2(10)
                          BYTE.
         stack 3(10)
                          BYTE.
         stack 4(10)
                          BYTE.
         stack 5(10)
                          BYTE.
         stack 6(10)
                          BYTE.
         stack 7(10)
                          BYTE)':
```

```
decode: PROCEDURE (estate387 ptr,errors387) EXTERNAL;
   DECLARE estate387 ptr. POINTER;
   DECLARE errors387
                           HWORD:
END decode:
encode: PROCEDURE (estate387 ptr, errors387,
                  retry control, retry flag) EXTERNAL;
   DECLARE estate387 ptr. POINTER;
   DECLARE errors387.
                           HWORD:
   DECLARE retry control, HWORD;
   DECLARE retry flag,
                           BYTE:
END encode:
/*** Trap handler is an 80387 exception handler
     template in PL/M-386 that calls the routines of
     EH387N.LIB. It assumes that the only unmasked
     exception is I, but shows where code for other
     exceptions could go.
***/
trap handler: PROCEDURE INTERRUPT REENTRANT PUBLIC;
   DECLARE estate387 STRUCTURE (
             operation
                           HWORD,
             argument
                           BYTE,
             arg1(5)
                           HWORD.
             argl full
                           BYTE.
             arg2(5)
                           HWORD.
             arg2 full
                           BYTE,
             result
                           BYTE.
             res1(5)
                           HWORD.
             res1 full
                           BYTE.
             res2(5)
                           HWORD.
             res2 full
                           BYTE.
             format
                           BYTE.
             register
                           BYTE.
             trap387env
                           Env387,
            trap387stack
                           Stack387):
 DECLARE retry control
                           HWORD;
 DECLARE retry flag
                           BYTE:
 DECLARE errors387
                           HWORD:
 DECLARE control387
                           HWORD:
```

```
/*** Any other variables you use must be declared
     inside the procedure. Because trap handler is
     REFNIRANT, variables are placed on the stack.
/*** Save the exception that caused the interrupt.
     GET$REAL$ERROR also clears the 80387 exceptions.
errors387 = GET$REAL$ERROR:
CALL decode (@estate387.errors387):
/*** Make control word when exception occurred the
     default for an encode retry operation and set
     retry flag. Unless you change the value of
     retry flag later, the subsequent call to encode
     will cause a retry of the operation.
 ***/
control387 = estate387.trap387env.control word;
retry flag = True;
/*** Mask the I exception bit in the control word ***/
retry control = retry control OR I$error$bit;
/*** Test to see if I exception bit was set. ***/
IF (errors387 AND I$error$bit) THEN DO;
/*** Put customized code for I exception here. ***/
END:
/*** You can put code to check errors387 for other
     exceptions and to customize error recovery next.
 ***/
ELSE DO:
END:
CALL encode (@estate387,errors387,
             retry control, retry flag);
END trap handler:
END handler module;
```



The 80387 implements the *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985, listed in the Preface. All of the 80386-based 80387 Support Libraries use the 80387 and this Standard as a guide for accuracy and performance.

## C.1 Decimal Conversion Library

The Standard specifies requirements for the accuracy of decimal conversions of 32-bit single and 64-bit double format numbers. Since DC387 uses 80-bit extended format arithmetic to perform its conversions, it exceeds the Standard's specifications.

Table C-1 shows the range of single and double format numbers for which the Standard requires the best possible conversion accuracy both from decimal to binary and from binary to decimal.

Table C-1 Correctly Rounded Decimal Conversion Ranges Required

	Decimal to B	inary	Binary to Decimal		
Format	Significand	Exponent	Significand	Exponent	
Single	10 <sup>9</sup> - 1	13	10 <sup>9</sup> - 1	13	
Double	10 <sup>17</sup> - 1	27	10 <sup>17</sup> - 1	27	

Table C-1 shows the number of decimal digits for which the Standard requires that decimal-to-binary and binary-to-decimal conversions be correctly rounded to the last bit. For example, it requires totally accurate single format binary-to-decimal conversions for positive values between and including 1E-13 and 999999999E13. DC387 exceeds the requirements in Table C-1.

The maximum number of significant digits that can be preserved on an in-and-out decimal-binary-decimal conversion can be determined mathematically for all floating-point formats. Such in-and-out conversions are accurate to the following:

- 6 significant digits for 32-bit single format
- 15 significant digits for 64-bit double format
- 18 significant digits for 80-bit extended format

DC387 realizes the best possible in-and-out decimal-binary-decimal conversion accuracy for every 80387 floating-point format. For example, DC387 guarantees that an 18-digit decimal number will retain its value when converted into extended format and then back to decimal.

The minimum number of significant decimal digits that must be produced on an in-and-out binary-decimal-binary conversion can also be determined mathematically for all floating-point formats. Such in-and-out conversions require that at least:

- 9 decimal digits be produced for 32-bit single format
- 17 decimal digits be produced for 64-bit double format
- 21 decimal digits be produced for 80-bit extended format

DC387 realizes the best possible in-and-out binary-decimal-binary conversion accuracy for 80387 single and double format values. However, DC387 cannot guarantee that an extended format value will retain every bit when converted to decimal and then back to extended format. Such a degree of accuracy requires DC387 to perform arithmetic beyond extended precision, which is very slow. DC387's loss of accuracy is, at worst, less than 3 bits for binary-decimal-binary extended format conversions.

Table C-2 shows a wider range of numbers for which the Standard requires slightly less accuracy for conversions. For numbers in this wider range, the Standard allows a conversion error that is not to exceed the rounding error by more than 0.47 units in the destination's least significant digit. DC387 also exceeds the requirements in Table C-2.

Table C-2 Decimal Conversion Ranges Required

	Decimal to B	inary	Binary to Decimal		
Format	Significand	Exponent	Significand	Exponent	
Single	10 <sup>9</sup> - 1	99	109 - 1	53	
Double	10 <sup>17</sup> - 1	999	10 <sup>17</sup> - 1	340	

## C.2 Common Elementary Function Library

Although most of the CL387 functions are outside the scope of the Standard, these functions round results, perform arithmetic on infinities, operate with NaNs and report exceptions in a manner consistent with the Standard's requirements.

## C.3 Exception Handling Library

The DECODE routine collects all the exception trapping information that is supplied by the 80386/80387 and required by the Standard:

- The kind of operation being performed
- The exceptions that occurred during the operation
- The format of the destination
- The rounded results for Overflow, Underflow, and Precision exceptions, even if a result will not fit in the destination format
- The values of the operation's arguments for Invalid and Zerodivide exceptions



Table C-2 Decimal Conversion Ranges Required

	Decimal to Binary		Binary to Decimal		
Format	Significand	Exponent	Significand	Exponent	
Single	109 - 1	99	10 <sup>9</sup> - 1	53	
Double	10 <sup>17</sup> - 1	999	10 <sup>17</sup> - 1	340	

## C.2 Common Elementary Function Library

Although most of the CL387 functions are outside the scope of the Standard, these functions round results, perform arithmetic on infinities, operate with NaNs and report exceptions in a manner consistent with the Standard's requirements.

## C.3 Exception Handling Library

The DECODE routine collects all the exception trapping information that is supplied by the 80386/80387 and required by the Standard:

- The kind of operation being performed
- The exceptions that occurred during the operation
- The format of the destination
- The rounded results for Overflow, Underflow, and Precision exceptions, even if a result will not fit in the destination format
- The values of the operation's arguments for Invalid and Zerodivide exceptions



Table C-2 Decimal Conversion Ranges Required

	Decimal to B	inary	Binary to Decimal		
Format	Significand	Exponent	Significand	Exponent	
Single	10 <sup>9</sup> - 1	99	10 <sup>9</sup> - 1	53	
Double	10 <sup>17</sup> - 1	999	10 <sup>17</sup> - 1	340	

## **C.2 Common Elementary Function Library**

Although most of the CL387 functions are outside the scope of the Standard, these functions round results, perform arithmetic on infinities, operate with NaNs and report exceptions in a manner consistent with the Standard's requirements.

## C.3 Exception Handling Library

The DECODE routine collects all the exception trapping information that is supplied by the 80386/80387 and required by the Standard:

- The kind of operation being performed
- The exceptions that occurred during the operation
- The format of the destination
- The rounded results for Overflow, Underflow, and Precision exceptions, even if a result will not fit in the destination format
- The values of the operation's arguments for Invalid and Zerodivide exceptions



The 80387 supports four integer and three real number formats. Each format stores the least significant digits of every number in the low-order byte that corresponds to the lowest 80386 memory address.

## **D.1 Integer Formats**

Table D-1 summarizes the integer formats supported by the 80387.

Table D-1 Integer Formats

Format	Range		Precision . Fie	ids Indefinite
Name	approx		(bits)	Value
Word Integer	10 <sup>4</sup>	16 bits	* (15-0)	8000H
Short Integer	10 <sup>9</sup>	32 bits	* (31-0)	800000000H
Long Integer	10 <sup>19</sup>	64 bits	* (63-0)	8000000000000
Packed BCD Integer	10 <sup>18</sup>	18 digit	s Sign (79) Digits (71-0)	Bits 79-64 set

<sup>\*</sup> in Two's Complement

### **D.2 Real Formats**

Table D-2 summarizes the real formats supported by the 80387.

Table D-2 Real Formats

Format Number(s)	Range		Precision	Fields	Bi	t
Name	approx				in Field	
Single	10 <sup>±38</sup>	24 bits	Sign		31	
3			Expone	ent	30-23	
			Signific		* 22-0	
Double	10 <sup>±308</sup>	53 bits	Sign		63	
			Expone	ent	62-52	
			Signific	and	*51-0	
Extended	10 <sup>±4932</sup>	64 bits	Sign		79	
			Expone	ent	78-64	
			Signific		63-0	

<sup>\*</sup> An implicit integer bit is part of the single and double formats. Thus, the single and double formats have 24 and 53 bits of precision even through their significands have 23 and 52 bits, respectively. The extended format has an explicit integer bit stored in bit 63.

Table D-3 summarizes the parameters of these real formats.

Table D-3 Summary of Real Format Parameters

	Real Number Format				
Parameter	Single		Double	Extended	
Format width in bits	32	64	80		
P (bits of precision)	24	53	64		
Exponent width in bits	8	11	15		
Emax	+127	+1023		+16383	
Emin	-126	-1022	-16382		
Exponent bias (normalized)	+127	+1023		+16383	

Table D-4 summarizes certain special values in each of these real formats.

Table D-4 Real Zero, Infinity, and QNaN Indefinite Values

Special Value	Real Number Single	Format Double	Extended		
+0	00000000H	00H	00H		
-0	80000000H	800H	800H		
+∞	7F800000H	7FF00H	7FF800H		
∞	FF800000H	FFF00H	FFFF800H		
QNaN Indefinite	FFC00000H	FFF800H	FFFFC00H		



# Appendix E ASM386 Floating-Point Instructions

This appendix summarizes the ASM386 floating-point instructions in alphanumeric order and in a series of tables with the following columns:

### Opcode

This column lists the hexadecimal machine code for each instruction. It uses the following notation:

- +1 adds a digit in the range 0..7 to the opcode; is an index to the floating-point stack element that is an operand of the instruction.
  - followed by a digit denotes the reg field value of the opcode's ModRM byte.

### Floating-Point Instruction

This column lists each floating-point instruction as it would appear in an ASM386 program. It uses the following notation:

ST	is the top element of the floating-point stack (pushed
	last, popped first); ST is equivalent to ST(0).

ST(1) is any element of the floating-point stack; 1 is a digit in the range 0..7.

m prefix denotes a memory operand.

r suffix denotes a real operand in 32-bit single, 64-bit double, or 80-bit extended format.

d suffix denotes an integer operand in 80-bit binary coded decimal (BCD) format.

j suffix denotes an integer operand in 16-bit word, 32-bit short, or 64-bit long format.

by suffix denotes an operand whose length is measured in bytes.

### Excp. Code

This column lists the hexadecimal value returned by the EH387 DECODE routine to the OPERATION field of the ESTATE387 structure (see Chapter 4).

### **Numerics Exceptions**

This column lists exceptions that the 80387 can generate for each instruction. It uses the following notation:

- IS denotes the Invalid Stack exception. This occurs when an operation causes stack overflow (trying to push an operand onto a non-empty stack element) or stack underflow (trying to pop an empty stack element or to use an empty stack element as an operand).
  - I denotes the Invalid exception. This occurs when an operation causes indeterminate results (such as 0/0 or attempting to take the square root of a negative number).
- D denotes the Denormal exception. This occurs when an instruction attempts to operate on a denormal operand.
- Z denotes the Zerodivide exception. This occurs when an operation on finite operands produces, without overflow, an infinite result
- O denotes the Overflow exception. This occurs when the exponent of the rounded result is too large for the format of the destination.
- U denotes the Underflow exception. This occurs either when U is unmasked and a non-zero result (rounded as though its exponent were unbound) would be too small for the format of the destination or when U is masked and such a result (rounded to the destination format) is also inexact.
- P denotes the Precision exception. This occurs when the exact mathematical result did not fit in the result format (the result is rounded).

#### Effect of Instruction

This column contains a concise definition of the operation performed by each instruction.

	Opcode (Hex)	Floating-Point Instruction	Excp. Code	Numerics Exceptions	Effect of Instruction
	D9 F0	F2XM1	1CH	IS,I,D,U,P	$ST := 2^{ST} - 1$
	D9 E1	FABS	01H	IS	ST:=  ST
	DE C1	FADD	05H	IS,I,D,O,U,P	ST(1) := ST(1) + ST, pop old ST
	D8 C0+i	FADD ST,ST(i)	05H	IS,I,D,O,U,P	ST := ST + ST(i)
	DC C0+ <i>i</i>	FADD ST(i),ST	05H	IS,I,D,O,U,P	ST(i) := ST(i) + ST
	D8 /0	FADD m32r	05H	IS,I,D,O,U,P	ST := ST + m32r
	DC /0	FADD m64r	05H	IS,I,D,O,U,P	ST := ST + m64r
	DE C0+i	FADDP ST(i),ST	05H	IS,I,D,O,U,P	ST(i) := ST(i) + ST, pop
	DF /4	FBLD m80d	10H	IS	Push, ST : = <i>m80d</i>
	DF /6	FBSTP m80d	0FH	IS,I,P	m80d := ST, pop
	D9 E0	FCHS	02H	IS	ST := -ST
	9B DB E2	FCLEX			Clear exceptions after check for pending unmasked numerics
					errors
,	D8 D1	FCOM	03H	IS,I,D	Compare ST with ST(1)
	D8 D0+i	FCOM ST(i)	03H	IS,I,D	Compare ST with ST(i)
	D8 /2	FCOM m32r	03H	IS,I,D	Compare ST with m32r
	DC /2	FCOM m64r	03H	IS,I,D	Compare ST with m64r
	D8 D9	FCOMP	03H	IS,I,D	Compare ST with ST(1), pop
	D8 D8+ <i>i</i>	FCOMP ST(I)	03H	IS,I,D	Compare ST with ST(i), pop
	D8 /3	FCOMP m32r	03H	IS,I,D	Compare ST with m32r, pop
	DC /3	FCOMP m64r	03H	IS,I,D	Compare ST with m64r, pop
	DE D9	FCOMPP	03H	IS,I,D	Compare ST with ST(1), pop twice
	D9 FF	FCOS	27H	IS,I,D,U,P	ST := cos(ST)
	D9 F6	FDECSTP			Decrement stack_top pointer
	DE F9	FDIV	09H	IS,i,D,Z,O,U,P	ST(1) := ST(1) / ST, pop old ST

Opcode (Hex)	Floating-Point Instruction	Excp. Code	Numerics Exceptions	Effect of Instruction
DC F8+i	FDIV ST(i),ST	09H	IS,I,D,Z,O,U,P	ST(i) := ST(i) / ST
D8 F0+i	FDIV ST,ST(i)	09H	IS,I,D,Z,O,U,P	ST := ST / ST(I)
D8 /6	FDIV m32r	09H	IS,I,D,Z,O,U,P	ST := ST / m32r
DC /6	FDIV m64r	09H	IS,I,D,Z,O,U,P	ST := ST / m64r
DE <b>F</b> 8+ <i>i</i>	FDIVP ST(i),ST	09H	IS,I,D,Z,O,U,P	ST(i) := ST(i) / ST, pop
DE F1	FDIVR	0AH	is,i,D,Z,O,U,P	ST(1) := ST / ST(1), pop old ST
DC F0+i	FDIVR ST(i),ST	0AH	IS,I,D,Z,O,U,P	ST(i) := ST / ST(i)
D8 F8+i	FDIVR ST,ST(i)	OAH	IS,I,D,Z,O,U,P	ST := ST(i) / ST
D8 /7	FDIVR m32r	0AH	IS,I,D,Z,O,U,P	ST := m32r / ST
DC /7	FDIVR m64r	0AH	IS,I,D,Z,O,U,P	ST := m64r / ST
DE F0+i	FDIVRP ST(i),ST	0AH	IS,I,D,Z,O,U,P	ST(i) := ST / ST(i), pop
DD C0+i	FFREE ST(i)			Empty ST(i)
DE /0	FIADD m16j	05H	IS,I,D,O,U,P	ST := ST + m16j
DA /0	FIADD m32j	05H	IS,I,D,O,U,P	ST := ST + m32j
DE /2	FICOM m16j	03H	IS,I,D	Compare ST with m16j
DA /2	FICOM m32j	03H	IS,I,D	Compare ST with m32j
DE /3	FICOMP m16j	03H	IS,I,D	Compare ST with <i>m16j</i> , pop
DA /3	FICOMP m32j	03H	IS,I,D	Compare ST with <i>m32j</i> , pop
DE /6	FIDIV m16j	09H	IS,I,D,Z,U,P	ST := ST / m16j
DA /6	FIDIV m32j	09H	IS,I,D,Z,U,P	ST := ST / m32j
DE /7	FIDIVR m16j	0AH	IS,I,D,Z,O,U,P	ST := m16j / ST
DA /7	FIDIVR m32j	0AH	IS,I,D,Z,O,U,P	ST := m32j / ST
DF /0	FILD m16j	10H	IS	Push, ST := <i>m16j</i>
DB /0	FILD m32j	10H	IS	Push, ST := <i>m32j</i>
DF /5	FILD m64j	10H	IS	Push, ST := m64j
DE /1	FIMUL m16j	08H	IS,I,D,O,U,P	ST := ST * m16j
DA /1	FIMUL m32j	08H	IS,I,D,O,U,P	ST := ST * m32j
D9 F7	FINCSTP			Increment stack_top pointer

Opcode	Floating-Point	Ехср.	Numerics	Effect of
(Hex)	Instruction	Code	Exceptions	Instruction
9B DB E3	FINIT	4010		Initialize 80387 after
				check for pending unmasked numerics errors
DF /2	FIST m16j	0FH	IS,I,U,P	m16j := ST
DB /2	FIST m32j	0FH	IS,I,U,P	m32/ := ST
DF /3	FISTP m16j	0FH	IS,I,U,P	m16j := ST, pop
DB /3	FISTP m32j	0FH	IS,I,U,P	m32j := ST, pop
DF /7	FISTP m64j	0FH	IS,I,U,P	m64j := ST, pop
DE /4	FISUB m16j	06H	IS,I,D,O,U,P	ST := ST - m16j
DA /4	FISUB m32j	06H	IS,I,D,O,U,P	ST := ST - m32j
DE /5	FISUBR m16j	07H	IS,I,D,O,U,P	ST := m16j - ST
DA /5	FISUBR m32j	07H	IS,I,D,O,U,P	ST := m32j - ST
D9 C0+i	FLD ST(i)	10H	IS .	Push, ST := old ST(i)
D9 /0	FLD m32r	10H	IS,I,D	Push, ST := <i>m32r</i>
DD /0	FLD m64r	10H	IS,I,D	Push, ST := $m64r$
DB /5	FLD m80r	10H	IS	Push, ST := <i>m80r</i>
D9 E8	FLD1	13H	IS	Push, ST := +1.0
D9 /5	FLDCW m2by	-		Control word := $m2b$
D9 /4	FLDENV m14/28by			Environment := m14b or m28by
D9 EA	FLDL2E	15H	IS	Push, ST := log, (e)
D9 E9	FLDL2T	14H	IS	Push, ST := log, (10)
D9 EC	FLDLG2	17H	IS	Push, ST := $\log_{10}(2)$
D9 ED	FLDLN2	18H	IS	Push, ST := log (2)
D9 EB	FLDPI	16H	IS	Push, ST := $\pi$
D9 EE	FLDZ	19H	IS	Push, $ST := +0.0$
DE C9	FMUL	H80	IS,I,D,O,U,P	ST(1) := ST(1) * ST, pop old ST
D8 C8+1	FMUL ST,ST(i)	08H	IS,I,D,O,U,P	ST := ST * ST(I)
DC C8+i	FMUL ST(i),ST	08H	IS,I,D,O,U,P	ST(i) := ST(i) * ST
D8 /1	FMUL m32r	08H	IS,I,D,O,U,P	ST := ST * m32r
DC /1	FMUL m64r	08H	IS,I,D,O,U,P	ST := ST * m64r
DE C8+i	FMULP ST(I),ST	08H	IS,I,D,O,U,P	ST(i) := ST(i) * ST, pop

Opcode (Hex)	Floating-Point Instruction	Excp. Code	Numerics Exceptions	Effect of Instruction
DB E2	FNCLEX:			Clear exceptions without check for numerics errors
DB E3	FNINIT	<del>-</del>	,	Initialize 80387 without check for numerics errors
D9 D0	FNOP		,	No operation :
DD /6	FNSAVE m94/108by			m94/108by :=
·		·		machine_state without check for numerics errors
D9 /7	FNSTCW m2by			m2by := control_word without check for numerics errors
D9 /6	FNSTENV m14/28by		1.	m14/28by := environment without check for numerics errors
DF F0	FNSTSW AX	-		AX := status_word without check for numerics errors
DD /7	FNSTSW m2by	****		<pre>m2by := status_word without check for numerics errors</pre>
D9 F3	FPATAN .	20H	IS,I,D,O,U,P	ST(1) := arctan(ST(1) / ST), pop old ST
D9 F8	FPREM	21H	IS,I,D,U	ST := remainder( integer_chop (ST / ST(1)))
D9 F5	FPREM1	24H	IS,I,D,U	ST := remainder( integer_round(ST / ST(1)))
D9 F2	FPTAN	1FH	IS,I,D,U,P	Y / X := tan(ST), ST := Y, push, ST := X
D9 FC	FRNDINT	22H	IS,I,P	ST := round(ST)
DD /4	FRSTOR m94/108by		·	Machine_state := m94by or m108by

	Opcode (Hex)	Floating-Point Instruction	Excp. Code	Numerics Exceptions	Effect of Instruction
	9B DD /6	FSAVE m94/108by	-		m94/108by := machine_state after check for pending unmasked numerics errors
	D9 FD	FSCALE	1AH	IS,I,D,O,U,P	$ST := ST * 2^{ST(1)}$
	D9 FE	FSIN	26H	IS,I,D,U,P	ST := sin(ST)
	D9 FB	FSINCOS	25H	IS,I,D,U,P	ST := sin(ST), push, ST := cos(ST)
	D9 FA	FSQRT	0CH	IS,I,D,U,P	ST := square root(ST)
	DD D0+i	FST ST(i)	oFH	IS	ST(i) := ST
	D9 /2	FST m32r	<b>OFH</b>	IS,I,O,U,P	m32r := ST
	DD /2	FST m64r	0FH	IS,I,O,U,P	m64r := ST
	DD D8+i	FSTP ST(i)	0FH	IS	ST(i) := ST, pop
	D9 /3	FSTP m32r	0FH	IS,I,O,U,P	m32r := ST, pop
	DD /3	FSTP m64r	0FH	IS,I,O,U,P	m64r := ST, pop
, die	DB /7	FSTP m80r	0FH	IS	m80r := ST, pop
	9B D9 /7	FSTCW m2by	-		<pre>m2by := control_word after check for pending unmasked numerics errors</pre>
	9B D9 /6	FSTENV m14/28by			m14/28by :=
					environment after check for pending unmasked numerics errors
	9B DF F0	FSTSW AX	E-101		AX := status_word after check for pending unmasked numerics errors
	9B DD /7	FSTSW m2by	-		<pre>m2by := status_word after check for pending unmasked numerics</pre>
					errors
	DE E9	FSUB	06H	IS,I,D,O,U,P	ST(1) := ST(1) - ST, pop

Opcode (Hex)	Floating-Point Instruction	Excp. Code	Numerics Exceptions	Effect of Instruction
DC E8+ <i>i</i>	FSUB ST(i),ST	06H	IS,I,D,O,U,P	ST(i) := ST(i) - ST
D8 E0+i	FSUB ST,ST(i)	06H	IS,I,D,O,U,P	ST := ST - ST(i)
D8 /4	FSUB m32r	06H	IS,I,D,O,U,P	ST := ST - m32r
DC /4	FSUB m64r	06H	IS,I,D,O,U,P	ST := ST - m64r
DE E8+ <i>i</i>	FSUBP ST(i),ST	06H	IS,I,D,O,U,P	ST(i) := ST(i) - ST, pop
DE E1	FSUBR	07H	IS,I,D,O,U,P	ST(1) := ST - ST(1), pop
DC E0+i	FSUBR ST(i),ST	07H	IS,I,D,O,U,P	ST(i) := ST - ST(i)
D8 E8+i	FSUBR ST,ST(I)	07H	IS,I,D,O,U,P	ST := ST(i) - ST
D8 /5	FSUBR m32r	07H	IS,I,D,O,U,P	ST := m32r - ST
DC /5	FSUBR m64r	07H	IS,I,D,O,U,P	ST := m64r - ST
DE E0+ <i>i</i>	FSUBRP ST(i),ST	07H	IS,I,D,O,U,P	ST(i) := ST - ST(i), pop
D9 E4	FTST	04H	IS,I,D	Compare ST to +0.0
DD E1	FUCOM	23H	IS,I,D	Compare ST with ST(1)
DD E0+i	FUCOM ST(i)	23H	IS,I,D	Compare ST with ST(i)
DD E9	FUCOMP	23H	IS,I,D	Compare ST with
				ST(1), pop
DD E8+ <i>i</i>	FUCOMP ST(i)	23H	IS,I,D	Compare ST with ST(i),
DA 50	FUCOLARR	0011	10.1.5	pop
DA E9	FUCOMPP	23H	IS,I,D	Compare ST with ST(1), pop twice
9B	FWAIT	***		Alternate of WAIT
D9 E5	FXAM	0DH		Status word
				condition_bits:=
				classification of ST
D9 C9	FXCH	0EH	IS	Exchange ST and ST(1)
D9 C8+i	FXCH ST(i)	0EH	IS	Exchange ST and ST(i)
D9 F4	FXTRACT	0BH	IS,I,D,Z	Push, ST(1) :=
				ST_exponent_field, ST := ST significand
D9 F1	FYL2X	1DH	IS,I,D,Z,O,U,P	ST(1) := ST(1) *
				log, (ST), pop old ST
D9 F9	FYL2XP1	1EH	IS,I,D,U,P	ST(1) := ST(1) * log <sub>2</sub> (ST + 1), pop old ST

This appendix is a quick reference for the Decimal Conversion, Common Elementary Function, and Exception Handling routines.

### F.1 Parameters and Results for DC387 Routines

Parameters to the DC387 routines must be pushed onto the 80386 stack. Each pointer is a 6-byte far (16-bit segment, 32-bit offset) or a 4-byte near (32-bit offset) pointer; the segment part of a far pointer must be pushed as a 4-byte quantity because DC387 uses a 32-bit wide stack. Any push of a segment register in a USE32 segment automatically pushes 4 bytes. DC387 routines return the status of the conversion to AL and return results to the locations specified by their parameters. They pop their input parameters from the stack before transferring control back to the caller.

The following summarizes each DC387 routine with its pointer parameters and results:

mqcBIN\_DECLOW converts a binary number to a decimal string.
 Its single input parameter, ADCB\_PTR, accesses a 17-byte ADCB block with the following fields:

B_BUF_PTR (6 bytes)	To the binary input buffer's value
PRSCN (byte)	Code values specifying input format:
	0 for single format
	2 for double format
	3 for extended format
LNGTH (byte)	Ordinal specifying length in bytes (digits) of output field at D_BUF_PTR
D_BUF_PTR (6 bytes)	To the decimal output buffer with implicit decimal point after rightmost ASCII digit
SCALE (2 bytes)	Two's complement integer value of output's true base, exponent
SIGN (byte)	ASCII for $+$ , $-$ , or special input values: • for NaN, $+$ for $+\infty$ , $-$ for $-\infty$ , 0 for $+0$ , and $-$ for $-0$

 mqcDEC\_BIN converts a decimal string to a binary number. Its single input parameter, DCB\_PTR, accesses a 14-byte DCB block with the following fields:

B\_BUF\_PTR (6 bytes) To the binary output buffer

PRSCN (byte) Code values specifying output format:

0 for single format2 for double format3 for extended format

LNGTH (byte) Ordinal specifying number of input

characters to be read at D\_BUF\_PTR

D BUF PTR (6 bytes)

To the decimal input buffer with string of

**ASCII characters** 

mqcDECLOW\_BIN converts a decimal string to a binary value.
 Its single input parameter, ADCB\_PTR, accesses a 17-byte ADCB block with the following fields:

B BUF PTR (6 bytes) To the binary output buffer

PRSCN (byte) Code values specifying output format:

0 for single format2 for double format3 for extended format

LNGTH (byte) Ordinal specifying number of input digits

to be read at D\_BUF\_PTR

D\_BUF\_PTR (6 bytes) To the decimal input buffer with implicit

decimal point after rightmost ASCII digit Two's complement integer value of input

SCALE (2 bytes) Two's complement integer value of

string's true base<sub>10</sub> exponent

SIGN (byte) ASCII for + or -

 mqcDUBL\_XTND converts a binary number from 64-bit double format to 80-bit extended format. Its input parameters are the following:

DUBL\_PTR To the binary input value XTND PTR To the binary output value

 mqcSNGL\_XTND converts a binary number from 32-bit single format to 80-bit extended format. Its input parameters are the following:

SNGL\_PTR To the binary input value XTND\_PTR To the binary output value

 mqcXTND\_DUBL converts a binary number from 80-bit extended format to 64-bit double format. Its input parameters are the following:

XTND\_PTR To the binary input value
DUBL\_PTR To the binary output value

 mqcXTND\_SNGL converts a binary number from 80-bit extended format to 32-bit single format. Its input parameters are the following:

XTND\_PTR To the binary input value SNGL PTR To the binary output value

## F.2 Arguments and Results for CL387 Functions

Most arguments to the CL387 functions must be pushed onto the 80387 stack; these are floating-point values. The other argments to the CL387 functions are integer values that must be loaded into particular 80386 registers or pushed onto the 80386 stack.

Most CL387 functions return floating-point results to the 80387 stack. The CL387 real-to-integer conversion functions return results in certain 80386 registers.

Table F-1 summarizes the arguments, results, and operation of each CL387 real function in 80387 stack position notation.

Table F-1 Summary of CL387 Real Function Arguments and Results

Name Results and Arguments		Operation	
mqerACS	ST := Arccos(ST)	Arc cosine	
mqerASN	ST := Arcsin(ST)	Arc sine	
mqerATN	ST := Arctan(ST)	Arc tangent	
mqerCOS	ST := cos(ST)	cosine	
mqerCSH	ST := cosh(ST)	hyperbolic cosine	
mqerDIM	ST := max(ST(1)-ST, +0)	positive difference	
mqerEXP	$ST := e^{ST}$	exponential	
mqeriA2	AX := roundaway(ST)	to word integer	
mqerlA4	EAX := roundaway(ST)	to short integer	
mqeriA8	EDX_EAX:=roundaway(ST)	to long integer	
mqerIAX	ST := roundaway(ST)	to nearest integer	
mqerIC2	AX := chop(ST)	to word integer	
mqerIC4	EAX := chop(ST)	to short integer	
mqerIC8	EDX_EAX := chop(ST)	to long integer	
mqerICX	ST := chop(ST)	to nearest integer	
mqerlE2	AX := roundeven(ST)	to word integer	
mqerlE4	EAX := roundeven(ST)	to short integer	
mqerIE8	EDX_EAX:=roundeven(ST)	to long integer	
mqerIEX	ST := roundeven(ST)	to nearest integer	
mqerLGD	$ST := log_{10}(ST)$	common logarithm	
mqerLGE	ST := In(ST)	natural logarithm	
mqerMAX	ST := max(ST(1),ST)	maximum	
mqerMIN	ST := min(ST(1),ST)	minimum	
mqerMOD	$ST := (ST(1) \mod ST)$	modulus with sign of ST(1)	
mqerRMD	ST := (ST(1) rem ST)	IEEE remainder	
mqerSGN	ST := ST(1), ST's sign	x with sign of y	
mqerSIN	ST := sin(ST)	sine	
mqerSNH	ST := sinh(ST)	hyperbolic sine	
mqerTAN	ST := tan(ST)	tangent	
mqerTNH	ST := tanh(ST)	hyperbolic tangent	
mqerY2X	$ST := ST(1)^{ST}$	y to real or integer power	
mqerYl2	ST := STAX	y to word integer power	
mqerYI4	ST := ST <sup>EAX</sup>	y to short integer power	
mqerYl8	$ST := ST^{EDX}_{-EAX}$	y to long integer power	
mqerYIS	St:=ST(dword ptr SS:ESP)	y to short integer power	

Table F-2 summarizes the arguments, results and operation of each CL387 complex function in 80387 stack position notation.

Table F-2 Summary of CL387 Complex Function Arguments and Results

Name	Results and Arguments	Operation
mqerCABS	ST :=   (ST,ST(1))	magnitude
mqerCACH	(ST,ST(1)):=Arccosh(ST,ST(1))	hyperbolic Arc
cosine		
mqerCACS	(ST,ST(1)):=Arccos(ST,ST(1))	Arc cosine
mqerCASH	(ST,ST(1)):=Arcsinh(ST,ST(1))	hyperbolic Arc sine
mqerCASN	(ST,ST(1)):=Arcsin(ST,ST(1))	Arc sine
mqerCATH	(ST,ST(1)):=Arctanh(ST,ST(1))	hyperbolic Arc tangent
mqerCATN	(ST,ST(1)):=Arctan(ST,ST(1))	Arc tangent
mgerCC2C	$(ST,ST(1)):=(ST(2),ST(3))^{(ST,ST(1))}$	w <sup>z</sup>
mgerCC2R	$(ST,ST(1)):=(ST(1),ST(2))^{ST}$	z <sup>x</sup>
mgerCCl2	$(ST,ST(1)):=(ST,ST(1))^{AX}$	$z^{j}$ (j = word integer)
mgerCCl4	$(ST,ST(1)):=(ST,ST(1))^{EAX}$	z <sup>j</sup> (j = short integer)
mqerCCl8	$(ST,ST(1)):=(ST,ST(1))^{EDX\_EAX}$	$z^{j}$ ( $i = long integer$ )
mgerCCIS	(ST,ST(1)):=(ST,ST(1))(dword ptr SS:ESP)	z <sup>j</sup> (j = short integer)
mgerCCOS	(ST,ST(1)):=cos(ST,ST(1))	cosine
mgerCCSH	(ST,ST(1)):=cosh(ST,ST(1))	hyperbolic cosine
mgerCDIV	(ST,ST(1)):=(ST(2),ST(3)/ST,ST(1))	division
mgerCEXP	$(ST,ST(1)):=e^{(ST,ST(1))}$	exponential
mgerCEXP	(ST,ST(1)):=Ln(ST,ST(1))	natural logarithm
mgerCMUL	(ST,ST(1)):=(ST(2),ST(3)*ST,ST(1))	multiplication
mgerCPOL	(ST,ST(1)):=( ST,ST(1) ,Arctan(ST,ST(1))	polar conversion
mgerCPRJ	$(ST,ST(1)):=\langle \infty,0 \text{ with sign}(ST(1))$	infinite projection
mgerCR2C	$(ST,ST(1)):=ST(2)^{(ST,ST(1))}$	x <sup>z</sup>
mgerCREC	(ST,ST(1)):= (ST*cos(ST(1)),ST*sin(ST(1)))	rectangular conversion
mgerCSIN	$(ST,ST(1)):=\sin(ST,ST(1))$	sine
mgerCSNH	(ST,ST(1)):=sinh(ST,ST(1))	hyperbolic sine
mgerCSQR	(ST,ST(1)):=sqrt(ST,ST(1))	square root
mgerCTAN	(ST,ST(1)):=tan(ST,ST(1))	tangent
mgerCTNH	(ST,ST(1)):=tanh(ST,ST(1))	hyperbolic tangent

## F.3 Parameters and Results for EH387 Routines

Parameters to the EH387 routines must be pushed onto the 80386 stack; they must be aligned on 4-byte boundaries and padded in the high-order bytes as needed.

The EH387 DECODE routine returns its results to the ESTATE387 data structure. The ENCODE routine accesses this structure to restore the 80387 machine state before returning control to the exception handler. Both EH387 routines pop their input parameters from the stack on return to the caller.

This section summarizes the EH387 routines with their parameters. Pointer parameters are a 6-byte far (16-bit segment, 32-bit offset) or a 4-byte near (32-bit offset) pointer. A summary of the ESTATE387 data structure follows this summary of the EH387 routines:

DECODE fills ESTATE387 with information about the interrupted operation, its arguments or results, and the 80387 machine state when the interrupt occurred. It has two parameters:

ESTATE387 PTR To ESTATE387 allocated on the stack or

In memory

Uncleared 80387 Status Word from XCPTNS387

interrupted operation

ENCODE accesses ESTATE387, and either retries the operation or restores the 80387 machine state, depending on the value of RETRY FLAG set by the exception handler. It has four parameters:

ESTATE387 PTR To ESTATE387 allocated on the stack or

in memory

XCPTNS387 Uncleared 80387 Status Word from

interrupted operation

RETRY CONTROL 80387 Control Word, supplied by handler,

for an ENCODE retry of interrupted

operation

RETRY FLAG Boolean byte, supplied by handler; set to

call ENCODE for retry attempt, cleared to

call ENCODE to restore 80387 and exit to

handler

• ESTATE387 has the following fields:

**OPERATION (2 bytes)** 

Exception opcode of interrupted

operation

ARGUMENT (byte)

Boolean bits set for CL387 complex operation (bit 7) and argument pushed on 80387 stack (bit 3); also has two bit fields specifying 80387 format of arguments with codes:

0 - no operand

1 - ST (80387 stack top)

2 - ST(1)

3 - ST(i) where i in REGISTER

4 - (see FORMAT)

5 - extended format

6 - long integer

7 - BCD integer

ARG1(5) (5 words) ARG1\_FULL (byte) ARG2(5) (5 words) ARG2\_FULL (byte) RESULT (byte) Array (80 bits) for first argument
Boolean set if ARG1 has defined value
Array (80 bits) for second argument
Boolean set if ARG2 has defined value
Boolean bits set for pop stack twice (bit
7) or pop stack once (bit 3); also has two
bit fields specifying 80387 format of
results as for ARGUMENT

RES1(5) (5 words) RES1\_FULL (byte) RES2(5) (5 words) RES2\_FULL (byte) FORMAT (byte) Array (80 bits) for first result
Boolean set if RES1 has defined value
Array (80 bits) for second result
Boolean set if RES2 has defined value
Specifies 80387 format when ARGUMENT
or RESULT field has code 3 with:

0 - single format

1 - short integer2 - double format

3 - word integer

REGISTER (byte)

Specifies 80387 stack position when ARGUMENT or RESULT field has code 4 Storage for 80387 machine state, a 32-bit

80387 Environment followed by 80387

stack

**SAVE387 (108 bytes)** 

# F.4 80387 Exceptions and Exception Opcodes

The EH387 routines do not incur exceptions. However, a reentrant trap handler can call ENCODE to retry operations that continue to generate exceptions. Table F-3 summarizes possible exceptions for the DC387 and CL387 routines. See Appendix E for possible exceptions generated by ASM386 floating-point instructions.

Table F-3 Summary of Possible DC387 and CL387 Exceptions

Routine	Exceptions	Routine	Exceptions
mqcBIN DECLOW	I,D,P	mqerY2X	I,D,Z,O,U
mqcDEC_BIN	O,U,P	mqerYl2/4/8/\$	I,D,Z,O,U
mqcDECLOW_BIN	O,U,P	mqerCABS	I,D,O,U,P
mqcDUBL_XTND	D	mqerCACH	I,D,O,U
mqcSNGL_XTND	D	mqerCACS	I,D,O,U
mqcXTND_DUBL	I,O,U,P	mqerCASH	I,D,O,U
mqcXTND_SNGL	1,O,U,P	mqerCASN	I,D,O,U
mqerACS	I,D	mqerCATH	I,D,O,U
mqerASN	I,D	mqerCATN	I,D,O,U
mqerATN	I,D	mqerCC2C	I,D,Z,O,U
mqerCOS	I,D	mqerCC2R	I,D,Z,O,U
mqerCSH	I,D,O	mqerCCl2/4/8/S	I,D,Z,O,U
mqerDIM	I,D,O,P	mqerCCOS	I,D,O,U
mqerEXP	I,D,O,U	mqerCCSH	I,D,O,U
mqerlA2/4/8/X	I,D,P	mqerCDIV	I,D,Z,O,U
mqerIC2/4/8/X	I,D,P	mqerCEXP	I,D,O,U
mqerIE2/4/8/X	I,D,P	mqerCLGE	I,D,Z
mgerLGD	I,D,Z	mqerCMUL	I,D,O,U
mqerLGE	I,D,Z	mqerCPOL	1,D,O,U
mqerMAX	1,D	mqerCPRJ	l (IS only)
mqerMiN	I,D	mqerCR2C	I,D,Z,O,U
mqerMOD	I,D	mqerCREC	I,D,U
mqerRMD	I,D	mqerCSIN	I,D,O,U
mqerSGN	I,D	mqerCSNH	1,D,O,U
mqerSIN	I,D	mqerCSQR	I,D
mqerSNH	I,D,O	mqerCTAN	I,D,O,U
mqerTAN mqerTNH	I,D I,D	mqerCTNH	I,D,O,U

Each Decimal Conversion and Common Elementary Function routine listed in Table F-3 has an exception opcode.

If 80387 exceptions are unmasked and one of the DC387 or CL387 routines generates a trap, its exception opcode fills one of the following fields:

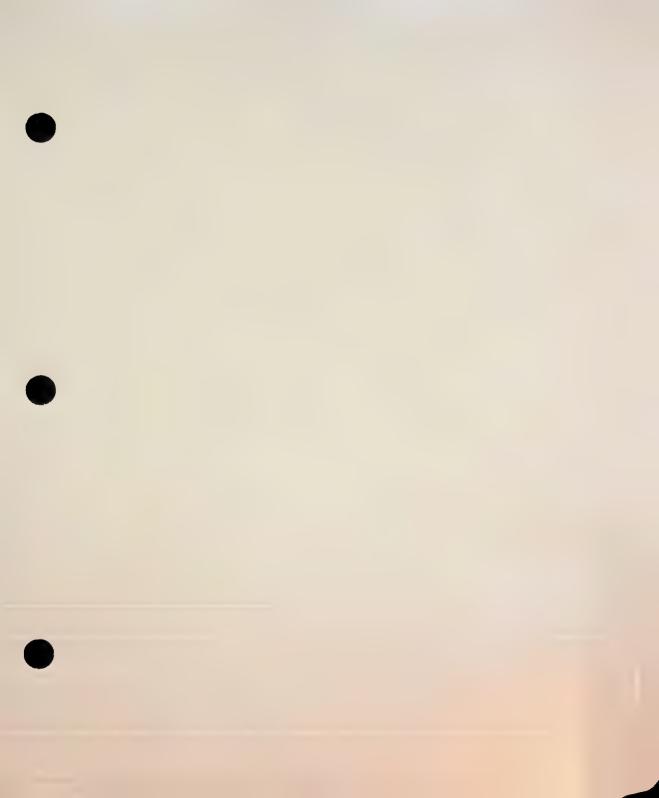
- Opcode in the 80387 Environment part of the machine state stored by the FSAVE/FNSAVE instructions
- OPERATION in the ESTATE387 structure after a call to the EH387 DECODE routine

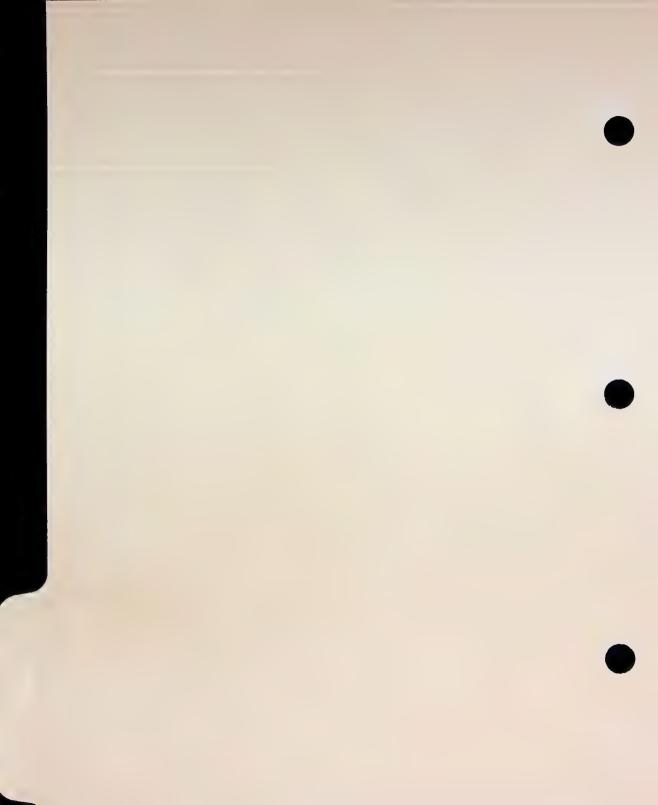
Table F-4 lists the hexadecimal exception opcodes for the Decimal Conversion and Common Elementary Function routines in numeric order.

Table F-4 DC387 and CL387 Exception Opcodes

Code	Routine	Code	Routine	Code	Routine	
D0H	mqcDEC BIN	17BH	mqerlE4	39AH	mqerCSNH	
D0H	mgcDECLOW BIN	17EH	mqerlA2	39BH	mqerCCOS	
D1H	mqcBIN_DECLOW	/ 17FH	mqerlC2	39CH	mqerCCSH	
D2H	mqcSNGL XTND	180H	mqerIE2	39DH	mqerCTAN	
D3H	mqcXTND_SNGL	184H	mqeriC8	39EH	mgerCTNH	
D4H	mqcDUBL_XTND	185H	mqerIA8	39FH	mqerCASN	
D5H	mqcXTND DUBL	186H	mqerlE8	3A0H	mqerCASH	
166H	mgerICX	264H	mgerSGN	3A1H	mgerCACS	
167H	mgerlAX	265H	mqerDIM	3A2H	mqerCACH	
168H	mqerlA4	269H	mqerMOD	3A3H	mqerCATN	
16BH	mgerEXP	26AH	mqerY2X	3A4H	mqerCATH	
16CH	mgerLGE	27AH	mqerRMD	3A7H	mqerCPRJ	
16DH	mqerLGD	27CH	mqerYI2	3A8H	mqerCPOL	
16EH	mqerSNH	27CH	mqerYI4	3A9H	mgerCREC	
16FH	mgerCSH	27CH	mqerYI8	58CH	mqerCMUL	
170H	mqerTNH	27CH	mqerYIS	58DH	mqerCDIV	
171H	mqerSIN	281H	mqerMiN	58EH	mqerCC2C	
172H	mqerCOS	282H	mqerMAX	591H	mqerCR2C	
173H	mqerTAN	283H	mqerCABS	592H	mqerCC2R	
174H	mgerASN	396H	mgerCSQR	<b>5</b> 95H	mqerCCl2	
175H	mqerACS	397H	mqerCLGE	595H	mqerCCi4	
176H	mqerATN	398H	mqerCEXP	595H	mqerCCl8	
178H	mqeriEX	399H	mgerCSIN	595H	mqerCCIS	
179H	mqeriC4		-			

The EH387 DECODE routine also returns exception opcodes for most ASM386 floating-point instructions to the OPERATION field of the ESTATE387 structure. See Appendix E for these codes.





This glossary defines many terms that have precise technical meanings as specified in the ANSI/IEEE Std 754-1985 IEEE Standard for Binary Floating-Point Arithmetic or as specified in this manual. Such terms are italicized in this glossary. You may therefore interpret any italicized terms or phrases as cross-references.

Base: (1) a term used in logarithms and exponentials. In both contexts, it is a number that is being raised to a power. The equations  $y = \log_b(x)$  and  $b^y = x$  where b is the base are equivalent.

Base: (2) a number that defines the representation being used for a string of digits. Base 2 is the binary representation; base 10 is the decimal representation; base 16 is the hexadecimal representation. In each case, the base is the factor of increased significance for each succeeding digit to the left.

Bias: a constant that is added to the true exponent of a real number to obtain the value in the exponent field of that number's floating-point representation in the 80387. To obtain the true exponent, subtract the bias from the field's value. For example, the single real format has a bias of 127 whenever the given exponent is nonzero. If the 8-bit exponent field contains 1000 0011 (131), the true exponent is 131 - 127 = +4.

Biased Exponent: the exponent as it appears in a floating-point representation of a real number. The biased exponent is interpreted as a nonnegative number.

Binary Coded Decimal: a method of storing numbers that retains a base<sub>10</sub> representation. Each decimal digit occupies 4 full bits (one hexadecimal digit). The hexadecimal values A through F (1010 through 1111) are not used. The 80387 supports a packed decimal format that consists of 9 bytes of binary coded decimal (18 decimal digits, each in the range 0..9) and one byte for the sign.

Binary Point: an entity just like a decimal point, except that it exists in binary numbers. Each binary digit to the left of the binary point is multiplied by an increasing power of two, beginning with 2<sup>0</sup>. Each binary digit to the right of the binary point is multiplied by an increasing negative power of two, beginning with 2<sup>-1</sup>.

Branch: a complex function which is analytic in some domain and which takes on one of the values of a multiple-valued function in that domain.

Branch Cut: a line or curve of singular points used in defining a branch of a multiple-valued complex function.

 $C_3...C_0$ : the four condition bits of the 80387 status word. These bits are set to certain values by some 80387 instructions and by certain CL387 and DC387 routines.

Characteristic: a synonym for the exponent field of a floating-point number.

Chop: to set one or more low-order bits of a number to zero, yielding the nearest representable number in the direction of zero.

CL387: the 80387 Common Elementary Function Library.

Complex Function: a function that operates on complex number(s).

Complex Number: a number that can be represented algebraically as z = x + iy where x and y are real numbers and  $i^2 = -1$ . A complex number can also be represented in rectangular or polar form.

DC387: the 80387 Decimal Conversion Library.

Denormal: a special form of floating-point number. On the 80387, a denormal is defined as a number that has a biased exponent of zero. By allowing a significand with leading zeros, the range of possible negative exponents can be extended by the number of bits in the significand. However, each leading zero is a bit of lost accuracy so this extended exponent range is obtained by reducing precision.

Double Extended: the Standard's term for the 80387's extended format. It consists of a sign, a 15-bit exponent biased by +16383, and an explicit integer bit in the 64-bit significand—a total of 80 explicit bits.

**Double Format:** a floating-point format supported by the 80387 that consists of a sign, an 11-bit exponent biased by +1023, and an implicit integer bit in the 52-bit significand—a total of 64 explicit bits.

EH387: the 80387 Exception Handling Library.

Emulator: a true software emulator performs floating-point operations exactly as if an 80387 were present in the system. However, the emulator's execution speed is slower.

Exception: any of the six conditions (invalid operation, denormal, zerodivide, numeric overflow, numeric underflow, and precision) detected by the 80387 and signaled by status flags (masked exceptions) or by status flags and traps (unmasked exceptions).

Exponent: (1) any number that indicates the power to which another number is raised.

Exponent: (2) the field of a *floating-point* number that indicates the number's order of magnitude. This would fall under the general definition (1) except that a *bias* must be subtracted to obtain the *floating-point* number's true exponent.

Extended Format: the 80387's implementation of the Standard's double extended format. Extended format is the main floating-point format used by the 80387. It consists of a sign, a 15-bit exponent biased by +16383, and a significand with an explicit integer bit and 63 fraction part bits—a total of 80 bits.

Far Library: 80387 Support Library modules 80387F.LIB, DC387F.LIB, CL387F.LIB, and EH387F.LIB are linked to programs located in different 80386 code segments from the segment named CODE32. Calls to far library routines assume access via a 6-byte effective address represented logically as a 2-byte selector (to the segment base address) plus a 4-byte offset from the segment base address.

Floating-Point: of or pertaining to a number that is expressed as a base, a sign, a signed exponent, and a significand. The value of the number is the signed product of its significand and the base raised to the power of the exponent. Floating-point representations are more versatile than integer representations in two ways. First, they include fractions. Second, their exponent parts allow a much wider range of magnitude than possible with fixed-length integer representations.

Fraction Part: the part of a floating-point significand that lies to the right of the binary point.

Gradual Underflow: a method of handling the numeric underflow exception that minimizes the loss of accuracy in the result. If there is a denormal number that represents the correct result, the 80387 returns that denormal if underflow is masked. Thus, digits are lost only to the extent of denormalization. Most computers return zero when underflow occurs, losing all significant digits.

Implicit Integer Bit: a part of the significand in the single and double real formats that is not explicitly stored. In these formats, only the fraction part is stored explicitly. The significand consists of the fraction part and an implicit integer bit to the left of the binary point that is always 1, except when the biased exponent is 0 (denormal with minimum biased exponent).

Indefinite: a special value that is returned by functions when the inputs are such that no sensible answer is possible. For each floating-point format there exists one quiet NaN that is designated the indefinite value. For each binary integer format, the negative number furthest from zero is considered the indefinite value  $(-2^{15}, -2^{31}, \text{ or } -2^{63})$ , as well as the minimum value. For the 80387 packed decimal format, the indefinite value contains all 1's in the uppermost byte (sign and next 7 bits).

Infinity: a value that has greater magnitude than any integer or real number. Infinity can be considered as another number that is subject to special 80387 rules of arithmetic. All three Intel floating-point formats provide representations for  $+\infty$  and  $-\infty$ .

Integer: a number (positive, negative, or zero) that is finite and has no fraction part. Integer can also mean the computer representation for such a number: a sequence of data bytes, interpreted in a standard way. Integers can be represented in floating-point format; this is what the 80387 does whenever an integer is pushed onto its stack.

Integer Bit: a part of the significand in floating-point formats. In these formats, the integer bit is the only part of the significand considered to be to the left of the binary point. The integer bit is usually 1 except when the biased exponent is 0 (denormal with minimum biased exponent). In the extended format, the integer bit is explicit; in the single and double formats the integer bit is implicit (not actually stored in memory).

Invalid Operation: the exception condition for the 80387 that covers all cases not covered by other exceptions. The invalid operation exception occurs for 80387 stack faults. It also occurs for certain arithmetic operations such as  $+\infty + -\infty$ , 0/0,  $\infty/\infty$ ,  $0^*\infty$ , taking the square root of a number less than zero, or taking a remainder x REM y where  $x = \infty$  or y = 0.

Long Integer: an integer format supported by the 80387 that consists of a 64-bit two's complement quantity.

Long Real: a non-Standard synonym for the 80387's double format.

Mantissa: a symonym for the fraction part of a floating-point number.

Masked: a term that applies to each of the six 80387 exceptions I,D,Z,O,U,P. If an exception is masked, the 80387 will not generate an interrupt when the exception condition occurs; it will provide a default result and set the exception status bit instead.

NaN: the Standard abbreviation for "Not a Number." A NaN is a bit pattern that does not represent any number, including infinity. The 80387 distinguishes between a signaling NaN (SNaN) and a quiet NaN (QNaN). It signals an invalid operation exception when an SNaN appears as the operand to an arithmetic operation or comparison. It creates a QNaN indefinite as its masked response to invalid operation exceptions.

Near Library: 80387 Support Library modules 80387N.LIB, DC387N.LIB, CL387N.LIB, and EH387N.LIB are linked to programs located in the CODE32 code segment; they share a combined data/stack segment named DATA. Calls to near library routines assume access via a 4-byte offset from the segment base address of CODE32.

Normal: the representation of a number in floating-point format in which the significand has an explicit or implicit integer bit equal to 1.

Normalize: to convert a denormal representation of the number to a normal representation.

Overflow: an exception condition in which the correct answer is finite, but it has magnitude too large to be represented in the destination format. This kind of overflow is also called "numeric overflow" to distinguish it from 80387 stack overflow, which causes the invalid operation exception.

Packed Decimal: an integer format supported by the 80387. A packed decimal number is a 10-byte quantity with nine bytes of 18 binary coded decimal digits and one byte for the sign bit.

Precision: (1) the effective number of bits in the significand of the floating-point representation of a number.

**Precision:** (2) an 80387 exception condition that results when a calculation does not return an exact answer. This exception is usually masked and ignored except by programmers who want to know when the least significant bit of any result has been obtained by rounding. The Standard calls the precision exception "inexact."

**Pseudodenormal:** one of a set of special values in *extended format*. The *exponent* field has all zeros, while the *significand* field has an explicit *integer bit* of 1. *Pseudodenormals* are not created by the 80387; they cause the D *exception* when encountered by the 80387 arithmetic instructions or by *CL387* functions.

Pseudoinfinity, Pseudo-NaN, and Pseudozero: see unsupported format

Quiet NaN (QNaN): a NaN in which the most significant bit of the fraction part of the significand is 1. Quiet NaNs as operands do not cause invalid operation exceptions except in ordered comparisons. They are intended to provide retrospective diagnostic information.

Real: any finite value (positive, negative, or zero) that can be represented by a possibly infinite decimal expansion. Reals can be represented as the points of a line marked off as a ruler. The term real can also refer to a floating-point number that represents a real value.

Riemann Sphere: A sphere whose south pole is tangent to the origin on the complex plane. Points on the Riemann sphere map one-to-one to points of the finite complex plane under stereographic projection. In other words, each point on the sphere except the north pole corresponds to the point on the plane at the end of the line segment from the north pole through that point on the sphere. The north pole's stereographic projection is the set of all points on the complex plane with at least one infinite component.

Short Integer: an integer format supported by the 80387 that consists of a 32-bit two's complement quantity. Note that short integer is not the shortest 80387 integer format—the 16-bit word integer is.

Short Real: a non-Standard synonym for the 80387's single format.

Signaling NaN (SNaN): a NaN that causes an invalid operation exception whenever it enters into an arithmetic operation or comparison (ordered or unordered).

Significand: the field of a floating-point number that, when multiplied by 2<sup>true\_exponent</sup>, yields the number's absolute value. The 80387 significand is composed of an integer bit and a fraction part. The integer bit is implicit in the 80387 single and double formats, but explicit in its extended format. The significand has an implicit binary point after the integer bit.

Single Extended: a floating-point format defined by the Standard: it provides greater precision than single format, and it has an explicit integer bit in the significand. The 80387's extended format meets the Standard's single extended and double extended requirements.

Single Format: a floating-point format, supported by the 80387, which consists of a sign bit, an 8-bit exponent biased by +127, an implicit integer bit, and a 23-bit significand—a total of 32 explicit bits.

Stack Fault: a special case of the *invalid operation* exception, which is indicated by S = 1 in the 80387 status word. This exception condition occurs due to stack overflow (trying to load to a non-empty stack element) or stack underflow (trying to pop an empty stack element or to use an empty stack element as an operand).

Standard: the IEEE Standard for Binary Floating-Point Arithmetic—ANSI/IEEE Std 754-1985.

Status Word: a 16-bit 80387 register that contains condition bits  $C_3...C_0$ , the 80387 stack pointer, busy, interrupt, and stack fault bits, and the exception byte (exception flags). The status word can be set or cleared directly by a program, but it is usually controlled by side effects to floating-point instructions.

Temporary Real (Tempreal): a non-Standard synonym for the 80387's 80-bit extended format.

Transcendental Constant: a number, such as  $\pi$  or e (2.71828182845904523536..) that cannot be represented as the root of an algebraic equation—in particular, it cannot be represented exactly as any rational number.

Transcendental Function: one of a class of functions for which polynomial formulas are approximations; results are seldom exact for more than isolated values. The exponential, logarithmic, trigonometric, and hyperbolic functions are transcendental.

Two's Complement: a method of representing integers. If the uppermost bit (sign) is 0, the number is considered positive and the rest of the bits represent the number's value. If the uppermost bit is 1, the number is negative: its value is obtained by subtracting (2<sup>bit\_count</sup>) from all the bits. For example, the 8-bit number 111111100 represents -4, obtained by subtracting 2<sup>8</sup> from 252.

Trap: an error detected by the 80386 and reported at the instruction boundary immediately after the instruction in which an exception was detected.

Trichotomy Law: Any number must be greater than, less than, or equal to another number.

Unbiased Exponent: the value of the exponent field of a floating-point number after subtraction of the format's bias. For example, if a single format exponent is 131, subtract the bias +127 to obtain the unbiased exponent +4; the real number being represented is (significand \* 2<sup>4</sup>).

Underflow: an exception condition in which the correct answer is nonzero, but the answer has a magnitude too small to be represented as a normal number in the destination floating-point format. The Standard specifies that an attempt be made to represent the number as a denormal, which can result in a loss of precision from the significand. This kind of underflow is also called "numeric underflow" to distinguish it from stack underflow, which causes the 80387 invalid operation exception.

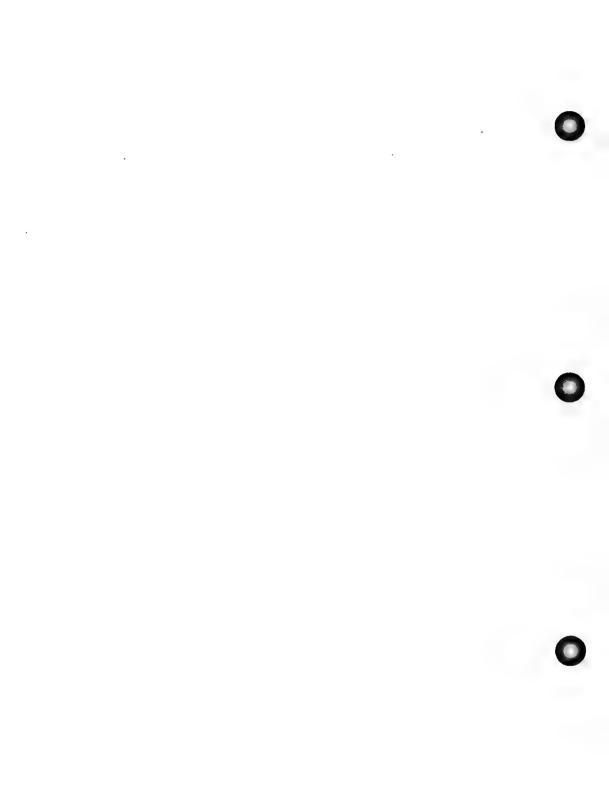
Unmasked: a term that applies to each of the six 80387 exceptions I,D,Z,O,U,P. If an exception is unmasked, the 80387 will generate an interrupt when the exception condition occurs so that a customized interrupt routine (a trap handler) can control recovery from the exception.

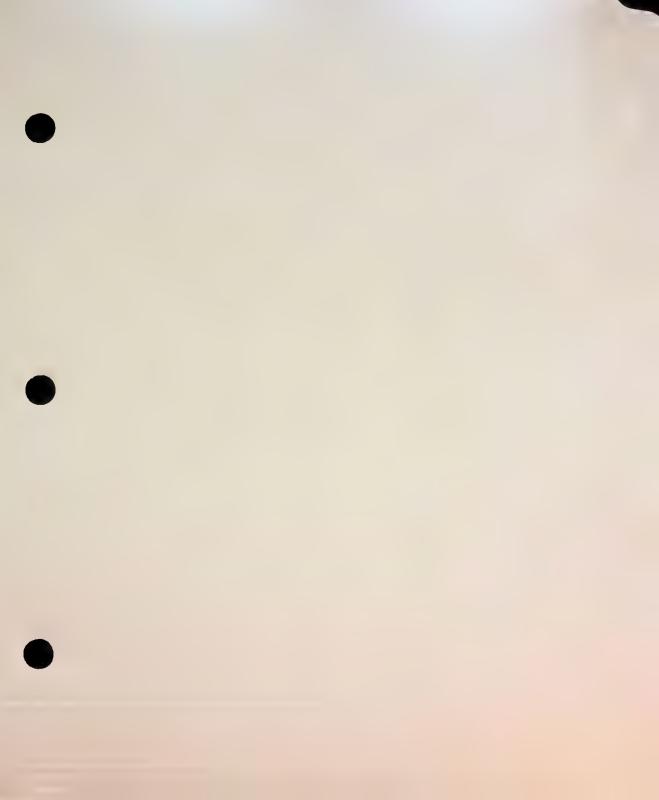
Unnormal: see unsupported format

Unsupported Format: any bit pattern that is not recognized by the 80387. This category includes formats that are recognized by the 8087 and 80287, namely: pseudo-NaN, pseudozero, pseudoinfinity, and unnormal.

Word Integer: an integer format supported by both the 80386 and 80387 that consists of a 16-bit two's complement quantity.

Zerodivide: an exception condition in which the operands are finite but the operation (not necessarily division) correctly produces a result of infinite magnitude.







For entries with many page numbers, see the page number in boldface first.

1POP bit, RESULT byte, 4-8 2POP bit, RESULT byte, 4-8 80387

data formats, D-1 initialization, 1-3 instructions, E-1 80387F.LIB, 1-2, 1-4 80387N.LIB, 1-2, 1-4

### A

Absolute value, complex number, 3-92, 3-96 Accuracy, decimal-binary conversions, C-1 to C-3 ADCB structure, 2-3, 2-4, 2-12, 2-20, F-1, F-2 AL register, 2-5, 2-7, 2-19, 2-33, F-1 Algebraic representation, complex number, 3-93 Alternative Decimal Conversion Block, see ADCB Angle, polar, 3-93 Arc cosine, 3-10, 3-13, 3-91, 3-103 Arc sine, 3-10, 3-15, 3-91, 3-111 Arc tangent, 3-10, 3-17, 3-91, 3-119 Arccos, see Arc cosine Arccosh, 3-91, 3-99 Arcsin, see Arc sine Arcsinh, 3-91, 3-107 Arctan, see Arc tangent Arctanh, 3-91, 3-115 ARG1 field, ESTATE387, 4-7, 4-11 ARGI FULL field, ESTATE387, 4-7, 4-11 ARG2 field, ESTATE387, 4-7, 4-11 ARG2 FULL field, ESTATE387, 4-7, 4-11 ARGUMENT byte, ESTATE387, 4-6, 4-11 ASCII digits, 2-12, 2-16, 2-20 ASM386 80387 instructions, E-1 ATYP1 and ATYP2 fields, ARGUMENT byte, 4-6 Augmented Decimal Conversion Block, see ADCB AX register, 3-27, 3-35, 3-43, 3-78, 3-129

```
Base 10 logarithm, see Logarithm, common
Base logarithm, see Logarithm, natural
BCD integer, D-1
Binary-binary conversions, 2-10, 2-24, 2-26, 2-28, 2-31, F-2, F-3
Binary-decimal conversion, 2-12, F-1
       C
Chop, 3-11, 3-35, 3-37, 3-39, 3-41, 3-59
Circular representation, see Polar representation
CL387, 1-1, 4-16, F-3
   PUBLIC symbols, A-2
   CL387F.LIB, 1-2, 3-2, A-1, B-7
   CL387N.LIB, 1-2, 3-2, A-1, B-7
CMX bit, ARGUMENT byte, 4-6
CODE32 segment, 1-2, 1-4, 2-2, 3-2, 4-2, 4-11
Common logarithm, see Logarithm, common
Complex
   division, 3-92, 3-147
  functions, 3-8, 3-90, 3-96 to 3-186, B-1, F-5
   infinity projection, 3-92
  magnitude, 3-92, 3-93, 3-96
  multiplication, 3-92, 3-158
   numbers, representations, 3-93, 3-161, 3-170
   plane, 3-92
  square root, 3-92, 3-96, 3-179
Control Word, 80387, 1-3, 2-3, 2-6, 2-9, 3-4, 3-7, 3-23, 4-1, 4-4, 4-11.
  4-18
Conversions
  binary-binary, 2-10, 2-24, 2-26, 2-28, 2-31, C-1, F-2, F-3
  complex numbers, 3-91, 3-161, 3-170
  decimal-binary, 2-10, 2-12, 2-16, 2-20, C-1, F-1, F-2
  in-and-out, C-2
  real to integer, 3-11, 3-27 to 3-49, F-3
Cosh, 3-10, 3-21, 3-91, 3-144
Cosine, 3-10, 3-19, 3-91, 3-141
CS
```

register, 4-12

selector, 80387 State, 4-12

D

```
Data formats, 80387, D-1
DATA segment, 1-2, 2-2, 2-6, 3-2, 3-3, 4-3, 4-11
DC387, 1-1, 4-16, F-1
  PUBLIC symbols, A-1
  DC387F.LIB, 1-2, 2-2, A-1, B-1
  DC387N.LIB, 1-2, 2-2, A-1, B-1
DCB structure, 2-3, 2-16, F-2
Decimal Conversion Block, see DCB
Decimal point, 2-16, 2-20
Decimal-binary conversions, 2-10, 2-12, 2-16, 2-20, F-1, F-2
DECODE, 4-1, 4-4, 4-9, 4-13, 4-16, 4-17, 4-18, B-10, C-3, F-6, F-10
Denormal
   argument, 4-16
   exception, 2-7, 3-5, E-3
Digits, ASCII, 2-12, 2-16, 2-20
Division, complex numbers, 3-92, 3-147
Domain, complex
   Arc cosine, 3-104
   Arc sine, 3-112
   Arc tangent, 3-120
   Arccosh, 3-100
   Arcsinh, 3-108
   Arctanh, 3-116
  cosh, 3-100
  cosine, 3-104
  natural logarithm, 3-155
  sine, 3-112
  sinh, 3-108
   tangent, 3-120
  tanh, 3-116
Double format real, C-1, D-2
DS register, 1-2, 2-6, 3-3, 4-3, 4-11
```

```
EAX register, 2-6, 3-3, 3-29, 3-31, 3-37, 3-39, 3-45, 3-47, 3-81, 3-84,
   3-132, 3-135, 4-3
EBP register, 2-6, 3-3, 4-3
EBX register, 2-6, 3-3, 4-3
ECX register, 2-6, 3-3, 4-3
EDI register, 2-6, 3-3, 4-3
EDX register, 2-6, 3-3, 3-31, 3-39, 3-47, 3-84, 3-135, 4-3
EFLAGS register, 4-10
EH387, 1-1, F-6
   PUBLIC symbols, A-3
   EH387F.LIB, 1-2, 4-2, 4-11, A-1
   EH387N.LIB, 1-2, 4-2, 4-11, A-1, B-8
EIP
  Offset, 80387 State, 2-9, 3-7, 4-12
  register, 4-12
ENCODE, 4-1, 4-4, 4-14, 4-18, 4-20, B-10, F-6, F-8
Environment, 80387, 2-8, 3-6, 4-9
ES register, 1-2, 2-6, 3-3, 4-3, 4-11
ESI register, 2-6, 3-3, 4-3
ESP register, 3-3
ESTATE387 structure, 4-4, 4-5, 4-11, 4-16, 4-18, B-10, F-7
Exception,
   byte, 80387, 2-5, 2-7, 2-19, 2-33, 3-4, 4-4, 4-16, 4-18
   handler, 3-3, 4-1, 4-10, 4-11, B-8
  opcode, 2-11, 3-12, 3-95, 4-16, E-3, F-10
  status, 80387, 2-8, 3-6
Exceptions, numeric, 2-7, 3-4, 4-5, 4-18, C-3, E-3, F-8
   ASM386 instructions, E-3
  CL387, F-8
  DC387, F-8
Exponential, 3-10, 3-25, 3-90, 3-151
Exponentiate, see Power functions
Extended format real, D-2
```

5

Far libraries, 1-2, 1-4, 2-2, 3-2, 4-2, B-5, B-8, F-1 FCOMP/FCOMPP instructions, 4-16 FCOS instruction, 3-19 Floating-point formats, D-2 instructions, E-1 FNCLEX instruction, 4-10, 4-13, 4-17 FNSTSW instruction, 4-10, 4-13, 4-17 FORMAT byte, ESTATE387, 4-8, 4-11 FPREM instruction, 3-59 FPREM1 instruction, 3-61 FPTAN instruction, 3-70 FS register, 2-6, 3-3, 4-3 FSAVE/FNSAVE instructions, 4-9, F-9 FSIN instruction, 3-66 FUCOMP/FUCOMPP instructions, 4-16 Functions, 2-5, see also Real functions and Complex functions

### G-H

GS register, 2-6, 3-3, 4-3
Hyperbolic functions, 3-10, 3-91
Arc cosine, see Arccosh
Arc sine, see Arcsinh
Arc tangent, see Arctanh
cosine, see Cosh
sine, see Sinh
tangent, see Tanh

1

IEEE754 Standard, 2-1, 3-9, C-1 Implicit integer bit, D-2 Indefinite values integer, D-1 ONaN. D-3 Inexact exception, see Precision exception Infinity complex projection, 3-92, 3-165 signed, 2-13, 2-28, 2-31, 3-8, D-3 INIT87, 1-3 INITFP, 1-3 Initialization library, 1-1 PUBLIC symbols, A-1 Installation, see the 80387 Support Library Release Notes Instruction pointer offset, see EIP offset Instruction pointer, 80387 State, 2-9, 3-7, 4-12 Instructions, ASM386, E-1 INT instruction, 4-15 Integer formats, D-1 Interrupt 16, 2-8, 3-6, 4-10 Interrupt handler, see Exception handler Invalid Operation exception, 1-3, 2-7, 3-5, E-3 IRETD instruction, 2-8, 3-6, 4-10, 4-15

### L

Linking to Support Libraries, 1-2, 3-2, 4-3, A-1 Ln, see Logarithm, natural Logarithm common, 3-10, 3-51 natural, 3-10, 3-53, 3-90, 3-154 Long integer, 3-31, 3-39, 3-47, 3-84, D-1 Long real, see Double format

Machine state, 80387, see State Machine pi, 80387, 3-9 Magnitude, complex number, 3-92, 3-93, 3-96, 3-161 Masked exceptions, 1-3, 2-7, 2-8, 3-4, 3-6 Maximum, 3-11, 3-55 Minimum, 3-11, 3-57 Modulus, 3-11, 3-59 mgcBIN DECLOW, 2-10, 2-12, B-2, F-1 mgcDEC BIN, 2-10, 2-16, F-2 mqcDECLOW\_BIN, 2-10, 2-16, 2-20, B-5, F-2 mqcDUBL\_XTND, 2-10, 2-24, F-2 mqcSNGL\_XTND, 2-10, 2-26, F-3 macXTND DUBL, 2-10, 2-28, F-3 macXTND SNGL, 2-10, 2-31, F-3 mgerACS, 3-10, 3-13, B-7 maerASN, 3-10, 3-15 mgerATN, 3-10, 3-17 mgerCABS, 3-92, 3-96 mgerCACH, 3-91, 3-99 mgerCACS, 3-91, 3-103 mgerCASH, 3-91, 3-107 mgerCASN, 3-91, 3-111 mqerCATH, 3-91, 3-115 mqerCATN, 3-91, 3-119 mqerCC2C, 3-90, 3-123 mgerCC2R, 3-90, 3-126 mqerCCI2, 3-90, 3-129 mqerCCI4, 3-90, 3-132 maerCCI8, 3-90, 3-135 mgerCCIS, 3-90, 3-138 mgerCCOS, 3-91, 3-141 mgerCCSH, 3-91, 3-144 mqerCDIV, 3-92, 3-147 mqerCEXP, 3-90, 3-151 mgerCLGE, 3-90, 3-154 mgerCMUL, 3-92, 3-158 mgerCOS, 3-10, 3-19 mqerCPOL, 3-91, 3-126, 3-161 mqerCPRJ, 3-92, 3-165 mgerCR2C, 3-90, 3-167 mgerCREC, 3-91, 3-170

mgerCSH, 3-10, 3-21 mgerCSIN, 3-91, 3-173 mgerCSNH, 3-91, 3-176 mgerCSQR, 3-92, 3-179 mgerCTAN, 3-91, 3-181 mgerCTNH, 3-91, 3-184 mgerDIM, 3-11, 3-23 mgerEXP, 3-10, 3-25 mgerIA2, 3-11, 3-27 mqerIA4, 3-11, 3-29 mgerIA8, 3-11, 3-31 mgerIAX, 3-11, 3-33 mgerIC2, 3-11, 3-35 mgerIC4, 3-11, 3-37 mgerIC8, 3-11, 3-39 mgerICX, 3-11, 3-41 mgerIE2, 3-11, 3-43, B-8 mgerIE4, 3-11, 3-45 mqerIE8, 3-11, 3-47 mgerIEX, 3-11, 3-49 mgerLGD, 3-10, 3-51 mgerLGE, 3-10, 3-53 mgerMAX, 3-11, 3-55 mqerMIN, 3-11, 3-57 mqerMOD, 3-11, 3-59 mqerRMD, 3-11, 3-61 mgerSGN, 3-11, 3-64 mgerSIN, 3-10, 3-66 mgerSNH, 3-10, 3-68 mgerTAN, 3-10, 3-70 mgerTNH, 3-10, 3-72 mgerY2X, 3-10, 3-74 mqerYI2, 3-10, 3-78 mgerYI4, 3-10, 3-81 mgerYI8, 3-10, 3-84 mgerYIS, 3-10, 3-87 Multiplication, complex numbers, 3-92, 3-158 Ν

NaN, 2-13, 2-28, 2-31, 3-8, D-3 Natural logarithm, see Logarithm, natural Near libraries, 1-2, 1-4, 2-2, 3-2, 4-2, B-2, B-7, B-8, F-1 Number formats, D-1 Numeric exceptions, see Exceptions

### 0

OMF386, 1-1, 2-1, 3-1, 4-3, B-1 op\_arg and op\_rad, 3-93, 3-161, 3-170 Opcode field, 80387 State, 2-9, 3-7, 4-12, F-9 Operand pointer, 80387 State, 4-12 selector, 80387 State, 4-12 OPERATION field, ESTATE387, 4-6, 4-11, F-9 Overflow exception, 2-8, 3-5, C-3, E-3

### P

Pi value, 80387 (machine\_pi), 3-9
Pointer parameters, 2-2 to 2-4, 2-11, 2-12, 2-16, 2-20, 2-24, 2-26, 2-28, 2-31, 4-4, 4-16, 4-18, B-2, B-5, F-1 to F-3, F-6
Polar representation, complex number, 3-91, 3-94, 3-161, 3-170
Positive Difference, 3-11, 3-23
Power functions, 3-10, 3-74 to 3-87, 3-90, 3-123 to 3-138, 3-167
Precision
exception, 2-8, 3-4, 3-6, C-3, E-3
mode, 1-3, 3-4, 3-23
Procedures, 2-5
Processor Extension Error, 80386, 2-8, 3-6, 4-10
Projection, infinite complex number, 3-92, 3-165
Protocols, exception handler, 4-10
PSH bit, ARGUMENT byte, 4-6
PUBLIC symbols, 1-1, 2-10, 3-8, 4-3, A-1

### Q-R

```
ONaN indefinite, 3-8, D-3
Raise to power, see Power functions
Range, complex
   Arc cosine, 3-104
   Arc sine, 3-112
   Arc tangent, 3-120
   Arccosh, 3-100
   Arcsinh, 3-108
   Arctanh, 3-116
   cosh, 3-100
  cosine, 3-104
   natural logarithm, 3-155
  sine, 3-112
  sinh, 3-108
   tangent, 3-120
  tanh. 3-116
Real
  formats, see Floating-point formats
  functions, 3-8, 3-13 to 3-89, F-4
  infinity values, D-3
  QNaN indefinite values, D-3
  conversions to integer, 3-11, 3-27 to 3-49
  zero values, D-3
Rectangular representation, complex number, 3-91, 3-93, 3-161, 3-170
Recursion, exception handler, 2-6, 3-3, 4-10, 4-19
Reentrancy, 2-6, 3-3, 4-3, 4-10, F-8
REGISTER byte, ESTATE387, 4-9, 4-11
Remainder, 3-11, 3-61
RES1 field, ESTATE387, 4-8, 4-11
RESI FULL field, ESTATE387, 4-8, 4-11
RES2 field, ESTATE387, 4-8, 4-11
RES2 FULL field, ESTATE387, 4-8, 4-11
Reserved names, see PUBLIC symbols
RESULT byte, ESTATE387, 4-7, 4-11
Riemann sphere, 3-165
```

#### Round

away from zero, 3-27, 3-29, 3-31, 3-33 real to integer, 3-11, 3-27 to 3-49, 3-61 to even, 3-43, 3-45, 3-47, 3-49, 3-61 toward zero, 3-35, 3-37, 3-39, 3-41 Rounding mode, 1-3, 2-6, 3-4, 3-9, 3-23 RTYP1 and RTYP2 fields, RESULT byte, 4-7, 4-8

### s

SAVE387 field, ESTATE387, 4-9 Segment names, 1-2, 1-4, 2-2, 3-2, 4-2, 4-11 SGN function, 3-11, 3-64 Short integer, 3-29, 3-37, 3-45, 3-81, 3-87, 3-132, 3-138, D-1 Short real, see Single format Signaling NaN (SNaN), 3-8 Sine, 3-10, 3-66, 3-91, 3-173 Single format real, C-1, D-2 Sinh, 3-10, 3-68, 3-91, 3-176 Special values, 80387, 2-13, 2-28, 2-31, 3-8, D-3 Square root, complex number, 3-92, 3-179 SS register, 1-2, 2-6, 3-3, 4-3, 4-11 Stack 80386, 1-2, 2-2, 2-5, 2-6, 3-3, 3-87, 3-138, 4-3, 4-4, 4-10, F-1, F-6 80387, 1-3, 2-6, 2-8, 2-9, 3-3, 3-6, 3-7, 4-3, 4-9, 4-16, B-1, F-3 frame, 80386, 4-11 overflow/underflow, 3-3, 3-6, E-3 State, 80387, 2-6 to 2-8, 3-6, 4-2, 4-4, 4-9, 4-16, 4-18, F-6 Status Word, 80387, 2-9, 3-4, 3-7, 3-59, 3-62, 4-1, 4-9 to 4-11 String, ASCII, 2-12, 2-16, 2-20

Tag Word, 80387, 2-9, 3-7, 4-11
Tangent, 3-10, 3-70, 3-91, 3-181
Tanh, 3-10, 3-72, 3-91, 3-184
Tempreal, see Extended format
Transcendental, see Trigonometric, Hyperbolic, Logarithm, and Exponential
Trap
80386, 3-6
handler, see Exception handler
Trigonometric functions, 3-10, 3-91
Truncate real, 3-11, 3-35, 3-37, 3-39, 3-41, 3-59

### U-V

Underflow exception, 2-8, 3-5, C-3, E-3 Unmasked exceptions, 2-8, 3-6, 4-1, 4-19, F-9 USE32, 1-2, 1-4, 2-2, 3-2, 4-2, 4-4, F-1 Values, indefinite, D-1, D-3

### W-Z

Word integer, 3-27, 3-35, 3-43, 3-78, 3-129, D-1 z\_re and z\_im, 3-93, 3-161, 3-170 Zero, signed, 2-13, 2-28, 2-31, 3-8, 3-23, D-3 Zerodivide exception, 3-5, C-3, E-3



### READER RESPONSE CARD

### We'd Like Your Opinion

Please use this form to help us evaluate the effectiveness of this manual and improve the quality of future versions.

To order publications, contact the Intel Literature Department (see page ii of this manual).

POOR		F	WERAGE			EXCELLENT
	ions would you l	racy material f ility of m JALITY category)	or your needs aterial OF THIS MA	ANUAL ain here:		
send you a you would like ur phone numb	0% off on the ne 50%-off certific us to call you for eer below.	ext Intel potate.	pecific sugge	ou buy. Send	t this boo	comments, and we'll ok, please additionally fi

Thanks for taking the time to fill out this form.



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

## **BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 79 HILLSBORO, OR

POSTAGE WILL BE PAID BY ADDRESSEE

DEVELOPMENT TOOLS OPERATION HF2-38 INTEL CORPORATION 5200 NE ELAM YOUNG PARKWAY HILLSBORO OR 97124-9978



Please fold here and close the card with tape. Do not staple.

Send your problem report and any additional material to the address printed above.

If you are in the United States and are sending only this card, postage is prepaid.

If you are sending additional material or if you are outside the United States, please insert this card and any enclosures in an envelope. Send the envelope to the above address, adding "United States of America" if you are outside the United States.

Thanks for your comments.



## International Sales Offices

BELGIUM Intel Corporation SA Rue des Cottages 65 B-1180 Brussels

DENMARK Intel Denmark A/S Glentevej 61-3rd Floor dk-2400 Copenhagen

ENGLAND Intel Corporation (U.K.) LTD. Piper's Way Swindon, Wiltshire SN3 1RJ

FINLAND Intel Finland OY Ruosilante 2 00390 Helsinki

FRANCE Intel Paris 1 Rue Edison-BP 303 78054 St.-Quentin-en-Yvelines Cedex

ISRAEL Intel Semiconductors LTD. Atidim Industrial Park Neve Sharet P.O. Box 43202 Tel-Aviv 61430

ITALY Intel Corporation S.P.A. Milandfiori, Palazzo E/4 20090 Assago (Milano) JAPAN Intel Japan K.K. 5-6 Tokodai, Tsukuba-shi Ibaraki, 300-26

NETHERLANDS Intel Semiconductor (Nederland B.V.) Alexanderpoort Building Marten Meesweg 93 3068 Rotterdam

NORWAY Intel Norway A/S P.O. Box 92 Hvamveien 4 N-2013, Skjetten

SPAIN Intel Iberia Calle Zurbaran 28-IZQDA 28010 Madrid

SWEDEN Intel Sweden A.B. Dalvaegen 24 S-171 36 Solna

SWITZERLAND Intel Semiconductor A.G. Talackerstrasse 17 8125 Glattbrugg CH-8065 Zurich

WEST GERMANY Intel Semiconductor GmbH Seidlestrasse 27 D-8000 Muenchen 2



For service or assistance with Intel products, call:

- 1-800-INTEL-4-U (1-800-468-3548) in the United States and Canada
- Your local Intel sales office in Europe or Japan
- Your Intel distributor in any other area

Intel stands behind its products with a world-wide service and support organization. If you have problems, need assistance, or have a question, Intel can provide:

- On-site or carry-in service for hardware products
- Phone support for all Intel products
- On-site consulting for designing with Intel products or using Intel products in your designs
- Customer training workshops
- Updates to software products

To get more information on these services or to take advantage of them, call the INTEL-4-U number above.

All Intel products have a minimum warranty of 90 days, and all warranties include one or more of the services listed above. Talk with your Intel salesperson or call the INTEL-4-U number to determine the warranty services available for this product and how to register for them.

Intel is committed to continuing service for all its products.



INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95052-8130 (408) 987-8080

Printed in U.S.A.

DEVELOPMENT TOOLS AND SOFTWARE